Meshfree and partition of unity methods for the analysis of composite materials

Barbieri, Ettore

Award date:
2010

Awarding institution:
University of Bath

Meshfree and Partition of Unity Methods for the Analysis of Composite Materials

Ettore Barbieri

Department of Mechanical Engineering

University of Bath

Thesis submitted as partial fulfilment for

the degree of Doctor of Philosophy

May 2010

To my loving family

Acknowledgements

First and foremost, I am grateful to my supervisor, Dr Michele Meo, for having believed in me and supported me during these three years at the University of Bath. Throughout these years, he has been a friend to me, rather than an academic.

He allowed me all the freedom I wanted, to follow my own steps. If today I can claim to be an independent researcher, it is because of him. His guidance was precious, especially in redirecting me on the right path, yet without limiting my creativity. I will always remember our interminable meetings, often over nice cups of Italian coffee.

Besides my supervisor, I would like to thank the academic and administrative staff of the Department of Mechanical Engineering, particularly those in the Materials Research Centre such as Dr Martin Ansell and Prof Darryl Almond.

My officemates and numerous friends played an important role in supporting me with their friendship in the many difficulties of the PhD's journey: I would like to thank (not in order of importance) my officemates Francesco Amerini, Francesco Ciampa, Stefano Angioni, Simon Pickering, Fulvio Pinto, Rocco Lupoi, Umberto Polimeno, Nando Dolce, Giovanni De Angelis, Mikael Amura, Tiemen Postma, Amit Visrolia and Panos Anestiou.

Abstract

The proposed research is essentially concerned on numerical simulation of materials and structures commonly used in the aerospace industry. The work is primarily focused on the study of the fracture mechanics with emphasis to composite materials, which are widely employed in the aerospace and automotive industry.

Since human lives are involved, it is highly important to know how such structures react in case of failure and, possibly, how to prevent them with an adequate design.

It has become of primary importance to simulate the material response in composite, especially considering that even a crack, which could be invisible from the outside, can propagate throughout the structure with small external loads and lead to unrecoverable fracture of the structure. In addition, structures made in composite often present a complex behaviour, due to their unconventional elastic properties.

A numerical simulation is then a starting point of an innovative and safe design.

Conventional techniques (finite elements for example) are not sufficient or simply not efficient in providing a satisfactory description of these phenomena. In fact, being based on the continuum assumption, mesh-based techniques suffer of a native incapacity of simulating discontinuities.

Novel numerical methods, known as Meshless Methods or Meshfree Methods (MM) and, in a wider perspective, Partition of Unity Methods (PUM), promise to overcome all the disadvantages of the traditional finite element techniques.

The absence of a mesh makes MM very attractive for those problems involving large deformations, moving boundaries and crack propagation.

However, MM still have significant limitations that prevent their acceptance among researchers and engineers. Because of the infancy of these methods, more efforts should be made in order to improve their performances, with particular attention to the computational time.

In summary, the proposed research will look at the attractive possibilities offered by these methods for the study of failure in composite materials and the subsequent propagation of cracks.

Contents

1 Statement of the problem **1**
1.1 The Need for Meshless Methods 1
1.2 Main differences between Finite Elements and Meshfree 8
1.3 Objectives 14
1.4 Outline of the thesis 16

2 Literature Review **17**
2.1 Review Papers and Books on Meshfree Methods 17
2.2 Classification of Meshfree Methods 18
2.3 Origins of Meshless Methods 20
2.4 Reproducing Kernel Methods and Element Free Galerkin 22
2.4.1 Reproducing Kernel Method 22
2.4.2 Element Free Galerkin 24
2.5 Applications to Fracture Mechanics 25
2.6 Applications to Nonlinear Mechanics 32
2.7 Other Meshfree Methods 35
2.8 Shortcomings, Recent Improvements and Trends 35

3 Description of the Method **39**
3.1 Definitions 39
3.1.1 Kernel or Weight Function 40
3.1.2 Window function and its derivatives 43
3.1.3 Partition of Unity and Consistency 46
3.2 Smooth Particle Hydrodynamics (SPH) 47
3.3 Reproducing Kernel Method (RKM) 47

3.4 Reproducing Kernel Particle Method (RKPM) 49
3.5 Moving Least Squares (MLS) 50
3.6 Discontinuities in Approximations 52
3.7 Intrinsic Enrichments 55
3.8 Extrinsic Enrichment 56
3.8.1 Extrinsic Global Enrichment 57
3.8.2 Extrinsic Local Enrichment 58
3.9 Element-Free Galerkin (EFG) 62

4 Improvements to the Method 66
4.1 *kd*-Trees for Neighbour Search 68
4.1.1 Speed-up using a background mesh: a mixed *kd-tree-elements* approach 75
4.1.2 The `Matlab` *kd*-tree 75
4.1.3 Results 75
4.2 The Moment Matrix 76
4.2.1 The window *function handle* 78
4.2.2 Computation of scaled coordinates 78
4.2.3 Computation of the polynomial basis 79
4.2.4 Computation of the sum of the polynomials 80
4.2.5 Derivatives of the moment matrix 82
4.2.6 Explicit Inverse of the Moments Matrix 83
4.3 Inversion of the Moment Matrix through Partitioning 84
4.3.1 Fast Inversion of the Moment Matrix and *p*-adaptivity . . 85
4.3.1.1 First Order Derivatives 89
4.3.2 Fast Inversion of the Moment Matrix and *h*-adaptivity . . 91
4.4 The Kernel Corrective Term 92
4.5 Assembly of the Stiffness Matrix 94
4.6 Results 99

5 Description of the Codes 108
5.1 The object-oriented programming 108
5.2 The class `RKPM` 109
5.2.1 The properties 109

5.2.2 The methods . . . 111
5.2.3 Inversion of the moments matrix through partitioning . . . 113
5.3 The class `Mesh` . . . 117
5.3.1 The properties . . . 117
5.3.2 The methods . . . 118
5.4 The class `Load` . . . 119
5.4.1 The properties . . . 120
5.4.2 The methods . . . 120
5.5 The class `Constraint` . . . 121
5.5.1 The properties . . . 121
5.5.2 The methods . . . 122
5.6 Other classes . . . 122
5.7 The class `LinearElastic2D` . . . 122
5.7.1 The properties . . . 123
5.7.2 The methods . . . 123
5.8 The class `LinearElastic3D` . . . 125

6 Application to Composite Materials: Delamination 130
6.1 Review of simulation of delamination . . . 133
6.2 Cohesive Zone Model . . . 137
6.3 The Cohesive Penalty approach . . . 142
6.3.1 The equations of equilibrium . . . 142
6.3.2 Discretization of equations of equilibrium . . . 144
6.3.3 Tangent Stiffness Matrix . . . 147
6.3.4 Convergence Issues in the Numerical Continuation . . . 148
6.3.5 The class `CohesiveContact` . . . 150
6.3.5.1 The properties . . . 150
6.3.5.2 The methods . . . 150
6.4 The Cohesive Segments method . . . 152
6.4.1 RKPM Enriched Approximation . . . 152
6.4.2 The equations of equilibrium . . . 155
6.4.3 Discretization of equations of equilibrium . . . 158
6.4.4 Tangent Stiffness Matrix . . . 161

6.4.5 Selection of Enriched Nodes . . . 163
6.4.6 Numerical Examples . . . 166
6.4.6.1 Double Cantilevered Beam: T300/977-2 . . . 170
6.4.6.2 Double Cantilevered Beam: XAS-913C . . . 172
6.4.6.3 Double Cantilevered Beam: AS4/PEEK . . . 173
6.4.6.4 Double Cantilevered Beam: Plane Stress or Plane Strain? . . . 173
6.4.6.5 End Loaded Split (ELS) . . . 175
6.4.6.6 Mode II End Notched Flexure: AS4/PEEK . . . 179
6.4.6.7 Mode II End Notched Flexure: T300/977-2 . . . 179
6.4.6.8 Central delamination with no pre-crack . . . 182
6.4.6.9 Double Cantilevered Beam with multiple cavities 182
6.4.7 Comparison with the Penalty Approach . . . 192
6.4.8 Convergence issues . . . 194
6.4.8.1 Effects of the inter-laminar strength τ^{max} . . . 194
6.4.8.2 Effects of the critical strain energy release rate G_c 196
6.4.8.3 Effects of a coarse mesh . . . 197
6.4.9 The class `Cohesive` . . . 198
6.4.9.1 The properties . . . 198
6.4.9.2 The methods . . . 200

7 Conclusions and future works 201
7.1 Future works . . . 203

A Computation of the Integral Terms in Reproducing Kernel Methods 204
A.1 The Window Function w and its primitives H_n . . . 204
A.1.1 Non Compact Support Kernels . . . 206
A.1.2 Compact Support Kernels . . . 208
A.2 Moments Matrix in One Dimension . . . 210
A.3 Two-Dimensional Domains . . . 214
A.4 Three-Dimensional Domains . . . 217
A.5 Explicit Expression for Line Integrals . . . 220

B Arc Length Numerical Continuation **225**

B.1 Continuation of Equilibrium 226

B.1.1 Predictor Phase 226

B.1.2 Corrector Phase 228

B.2 Line Search algorithm 231

References **255**

List of Figures

1.1 Example of Mesh in Finite Element Analysis 2
1.2 Discretization: domain Ω is subdivided in elements Ω_E 3
1.3 Examples of failure in laminated composite 5
1.4 Example of Meshfree Method: nodes discretization of a femur . . 7
1.5 Discretization in Meshless Methods 8
1.6 Mesh Distortion in FE 11
1.7 Difference between FE and meshless shape functions 13

2.1 Visibility criterion 25
2.2 Plate with circular hole under remote tension 26
2.3 Typical Nodes Arrangement for edge crack problem 27
2.4 Diffraction criterion 28
2.5 h-adaptivity in FE (left) and meshfree (right) 30
2.6 Example of Nonlinear Analysis: box-beam under impact 33
2.7 Example of Large Deformation Analysis: hyperelastic material tensile test 33

3.1 Examples of compact support 40
3.2 Window function of the 2k-th order spline 43
3.3 Definition of the variables ξ and η 44
3.4 SPH shape functions 48
3.5 Weight function for the visibility criterion 53
3.6 Weight function for the transparency criterion 54
3.7 Branch functions 60
3.8 Near crack-tip polar coordinates 60

3.9 Quadrature methods in MM 64
3.10 Bounding box integration 65

4.1 Flowchart of the *classic* algorithm in meshfree methods 67
4.2 Sparsity pattern for meshfree shape functions on a segment 70
4.3 Neighbour Search: red crosses: evaluation points inside the support; blue crosses: evaluation points outside the support 71
4.4 Neighbour Search: example of the mapping in equation (4.8) . . . 72
4.5 Example of kd-tree 74
4.6 Computational times for neighbour search: comparison between brute force and *kd*-tree 76
4.7 Computational times for neighbour search using *kd*-tree 77
4.8 Flowchart of the proposed algorithm in meshfree methods 86
4.9 Three dimensional case: geometry 100
4.10 Quadrature cells for the three-dimensional case 101
4.11 Computational run-times for the inversion of the moment matrix . 105
4.12 Computational run-times for the shape functions 105

5.1 Three-dimensional code: vertical bending 127
5.2 Three-dimensional code: torsion 128
5.3 Three-dimensional code: lateral bending 129

6.1 Failure modes for internal delamination 133
6.2 Examples of equilibrium paths for non-linear equations 136
6.3 Failures in the load control and displacement control 137
6.4 Reference frame for the interface 138
6.5 Normal traction T_n - relative displacement u_n curve for $\alpha = 0$ (equation (6.3)) 139
6.6 Bilinear softening model 141
6.7 Description of the problem 143
6.8 Reference transformation from global coordinates to local coordinates 144
6.9 Cohesive segments method 152
6.10 Decomposition of a continuous crack in discrete cracks 153
6.11 Cohesive segments: Gaussian points defined over the segment . . 165

6.12 Cohesive segments: Gaussian points defined over the segment (zoom) 165
6.13 Cohesive segments: selection of enriched nodes 166
6.14 Cohesive segments: selection of enriched nodes (zoom) 166
6.15 Double Cantilevered Beam: geometry and boundary conditions . . 168
6.16 End Loaded Split: geometry and boundary conditions 168
6.17 End Notched Flexure: geometry and boundary conditions 168
6.18 Simply Supported Double Cantilevered Beam: geometry and boundary conditions 169
6.19 Central Delamination: geometry and boundary conditions 169
6.20 Delamination Stages for DCB test T300/977-2: transverse stress σ_y plot 171
6.21 Force-Displacement curve for DCB test T300/977-2 171
6.22 Force-Displacement curve for DCB test XAS-913C 172
6.23 Delamination stress plot for DCB test XAS-913C 173
6.24 Force-Displacement curve for DCB test AS4/PEEK 174
6.25 Force - Displacement curve for DCB test in Chen *et al.* (1999) . . 176
6.26 Force-Displacement curve for ELS test 177
6.27 Delamination Stages for ELS test: longitudinal stress σ_x plot . . . 178
6.28 Delamination Stages for ELS test: longitudinal stress τ_{xy} plot . . 178
6.29 Force-Displacement curve for ENF test (AS4/PEEK) 180
6.30 Delamination Stages for ENF test: longitudinal stress σ_x plot . . 180
6.31 Delamination Stages for ENF test: shear stress τ_{xy} plot 181
6.32 Force-Displacement curve for ENF test (T300/977-2) 181
6.33 Delamination stages for the case of central delamination: longitudinal stress σ_x plot 183
6.34 Delamination stages for the case of central delamination: longitudinal stress σ_y plot 184
6.35 Delamination stages for the case of central delamination: longitudinal stress τ_{xy} plot 185
6.36 Force-Displacement curve for central delamination 186
6.37 Force-Displacement curve for central delamination (zoom on the initial phase) 186

6.38 Force-Displacement curve for DCB test T300 with coarser mesh 5×378 . . . 187
6.39 DCB with multiple cavities . . . 189
6.40 DCB with one cavity close to the crack-tip . . . 189
6.41 DCB with one internal cavity . . . 190
6.42 Force-Displacement curve for DCB with cavities . . . 190
6.43 Shear Stress τ_{xy} plot for DCB with pre-crack and multiple failures 191
6.44 Force-Displacement curve for DCB with one cavity close to the crack-tip . . . 191
6.45 Force-Displacement curve for DCB with one cavity close to the crack-tip . . . 192
6.46 Comparison between cohesive segments and cohesive contact: DCB 193
6.47 Comparison between cohesive segments and cohesive contact: DCB XAS-913C . . . 193
6.48 Comparison between cohesive segments and cohesive contact: ELS 194
6.49 Convergence issues: variable interlaminar strength . . . 195
6.50 Convergence issues for different G_c . . . 196
6.51 Bounce back of the continuation . . . 197

A.1 Non-Compact Support Kernel Functions: Gaussian Function . . . 207
A.2 Compact Support Kernel Functions: $2k-th$ order spline . . . 209
A.3 Moments Matrix in One Dimension . . . 211
A.4 Correction Factors in One Dimension for basis function $\mathbf{p}^T(x) = [1 \;\; x]$ . . . 212
A.5 First Derivative of Moments Matrix in One Dimension . . . 213
A.6 First Derivative of Correction Factors in One Dimension for basis function $\mathbf{p}^T(x) = [1 \;\; x]$ . . . 214
A.7 Four Nodes Quadrilateral element . . . 215
A.8 Examples of moment matrix entries in two dimensions . . . 216
A.9 Three-dimensional polygonal subdivision . . . 218

List of Tables

1.1 Main differences between FE and MM 9

4.1 Elastic Properties for the benchmark case 100
4.2 Geometry for the benchmark case 100
4.3 Computational run-times for the connectivity 102
4.4 Computational run-times for the correction term 102
4.5 Computational run-times for the assembly 103
4.6 Computational run-times for the moment matrix 103
4.7 Computational run-times for the inversion of the moment matrix . 103
4.8 Computational Run-times for the Shape Functions 104
4.9 Inversion of the moment matrix (classic method): goodness of quadratic fit $f(x) = p_1x^2 + p_2x + p_3$ 106
4.10 Inversion of the moment matrix (new method) : goodness of linear fit $f(x) = p_1x + p_2$ 106
4.11 Shape functions (classic method) : goodness of quadratic fit $f(x) = p_1x^2 + p_2x + p_3$ 107
4.12 Shape functions (new method) : goodness of linear fit $f(x) = p_1x+p_2$ 107

6.1 Properties for T300/977-2 170
6.2 Properties for DCB test XAS-913C 172
6.3 Properties for AS4/PEEK 174
6.4 Properties for material in Chen *et al.* (1999) 177
6.5 Properties for DCB test with cavities 188

Nomenclature

Acronyms

ALE Arbitrary Lagrangian Eulerian

CAD Computer Aided Design

CDM Continuum Damage Mechanics

CTM Cartesian Transformation Method

DCB Double Cantilevered Beam

EFG Element Free Galerkin

ELS End Loaded Split

ENF End Notched Flexure

FDM Finite Difference Method

FEM Finite Element Method

FPM Finite Point Method

FVM Finite Volume Method

LBIE Local Boundary Integral Equation

MFEM Meshless Finite Element Method

LS Line Search

MLPG Meshless Local Petrov-Galerkin

MLS Moving Least Squares

MLSSPH Moving Least Squares Smooth Particle Hydrodynamics

MLSRKM Moving Least Squares Reproducing Kernel Method

MM Meshfree Method

NEM Natural Element Method

NR Newton-Raphson scheme

OOP Object Oriented Programming

PDE Partial Differential Equation

PU Partition of Unity

RKM Reproducing Kernel Method

RKPM Reproducing Kernel Particle Method

SIF Stress Intensity Factor

SPH Smooth Particle Hydrodynamics

XFEM eXtended Finite Element Method

Chapter 1

Statement of the problem

1.1 The Need for Meshless Methods

Numerical methods are crucial for an accurate simulation of physical problems as the underlying partial differential equations usually need to be approximated. Approximation is necessary either for the complexity of the equations themselves or for the complexity of the geometry where these equations need to be solved.

Among all the available numerical schemes, mesh-based methods have become the most popular tool for engineering analysis over the last decades in academic and industrial applications. Conventional mesh-based numerical methods are Finite Element Method (FEM) and Finite Volume Method (FVM) among the most well-known members of the thoroughly developed methods. FEM for example is nowadays widely used by engineers in all fields and several well-assessed commercial codes are available. The applications of FEM range from structural mechanics, thermal analysis, acoustics, fluid dynamics, electromagnetism and even multi-physics (i.e. coupling of physical phenomena) simulations.

A vast amount of scientific literature has been produced in the recent years since their invention in 1960 Clough (1960). A large mathematical work has been done by researchers to prove their convergence for both fluid and solid mechanics. This testifies that FEM performs really well for a broad range of cases where a continuum can be easily subdivided (not broken) into discrete elements. This operation is usually called *discretization* (figure 1.2). Each element is basically made of a certain number of labelled nodes and a sequence of number (*the nodes'*

labels) that defines how these nodes are connected. This is the fundamental characteristic of the mesh-based methods: a priori definition of the nodes connectivity, commonly known as *mesh*. The mesh permits the compatibility of the interpolation (figure 1.1), meaning that the resulting approximation is continuous.

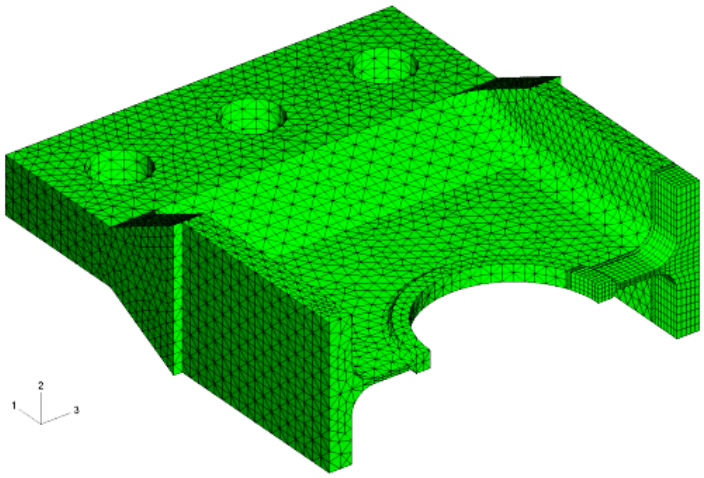


Figure 1.1: Example of Mesh in Finite Element Analysis

FEM is robust and in particular for mechanics has been thoroughly and continuously developed since its creation. This includes static and dynamic, linear or non-linear stress analysis of structures. The creation of a mesh, though, could represent a challenging task. In fact, the analyst spends most of his work-time in creating the mesh. It has become a major component of the total cost of a simulation project because currently mere hardware is exponentially decreasing its price. According to Liu (2003), who in 2003 wrote the first comprehensive book on meshfree methods, *the logical consequence is that the issue is now more the manpower time, and less the computer time. Consequently, in an ideal world, the meshing process would be fully performed by the computer without human intervention.* At the same time, with the increasing demand of high performance products in the manufacturing industry, the problems in computational mechanics grow more and more difficult and *mechanicians* need more and more sophisticated numerical tools. These tools will eventually allow engineers to design even more complex and high quality products. These tools are nowadays not purely

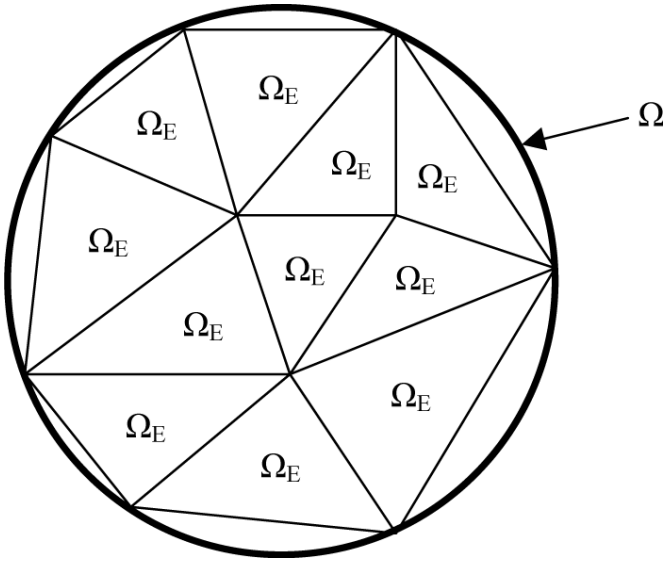


Figure 1.2: Discretization: domain Ω is subdivided in elements Ω_E

concern of academia, but can be very applicative indeed. Examples of such complicated physical phenomena that are of interest in the manufacturing industry are, for example, extrusion and molding, that require treatments of extremely large deformations, or the computations of castings, where it is critical to predict the propagation of interfaces between solids and liquids or to track the growth of phase boundaries.

Another perhaps more clear example is the simulation of propagation of cracks with arbitrary and complex paths. This is even more complicated when more and more heterogeneous materials are created as the frontiers of the material science continue to expand. Ultimately the frontiers would expand not only at a macroscopic level, but go deeper at the smallest length scales, bringing engineers, chemists and physicists to work together. This is for example what is currently happening for the case of the nanotechnology field.

The technological evolution is particularly evident for composite materials. Composite materials are the combination of two materials, generally a matrix that surrounds and binds together a cluster of fibers or fragments of a much stronger material (the reinforcement). There are many different types of composite materials: to mention only some of them, there exist carbon-fiber reinforced plastic, shape memory polymer composites, metal matrix composites, ceramic matrix

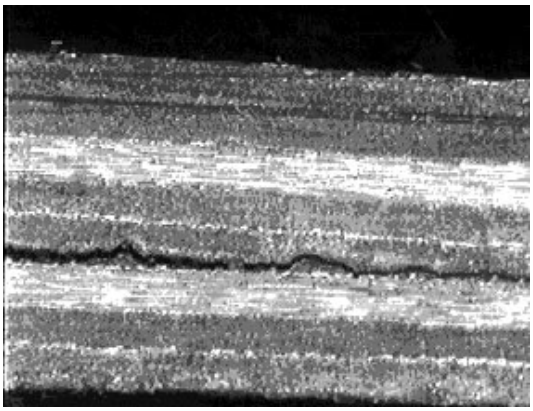
composites and sandwich composites. Being fairly complex materials, their failure cannot be easily treated with standard methods because failure modes for composite materials are indeed quite different from the ones in metals. For example, for laminated composite materials, where different plies are stacked up, a very important failure mechanism is *delamination* (figure 1.3a), due to the lack of reinforcement in the thickness. Delamination occurs for example in case of impact events or for manufacturing defects. Moreover, since each ply is a mixture of plastic matrix and fibers, fiber pull-out (figure 1.3b), fiber breakage (figure 1.3c), fiber-matrix debonding (figure 1.3d) and matrix cracking (figure 1.3e) may initiate and propagate, compromising the integrity of the structure.

However, their unique characteristics of high strength and stiffness combined to a much lower weight than aluminum and steel, make composite materials very attractive to those manufacturing industries like automotive and aerospace, where these requisites are more and more competitive values and influence the choice of the customers. It suffices to think about the fuel that can be saved with lighter vehicles without compromising the safety of the passengers. Less fuel would also mean a positive impact on the environment. It is not a coincidence that the two biggest aerospace industries like Boeing and Airbus have been trying (and still are) to build either primary structural parts or entire airplanes in composite. Another example is the automotive sector, one of the most competitive industrial markets, where faster and more performing cars are proposed every year. This is more stressed in competitive cutting edge sport sectors like Formula 1 when even the smallest details can make the difference in a race.

Therefore the full understanding of these complex materials is at the same time complicated but important, yet vital for a safe and competitive design. Their importance, from an academic point of view, justifies the number of research centers on composite materials and the number of researchers involved in the many aspects of composite materials. The last international and most important conference on composite materials [1] was attended by thousands of scholars from all over the world.

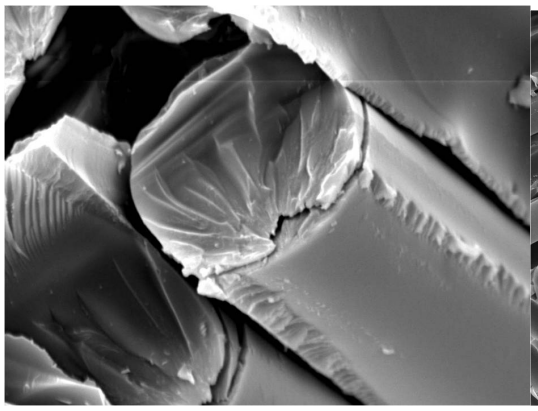
We might then want to ask ourselves, what are the difficulties in the simulation of the mechanics of composite materials? Delaminations and fracture

[1] International Conference on Composite Materials 17 Edinburgh July, 2009

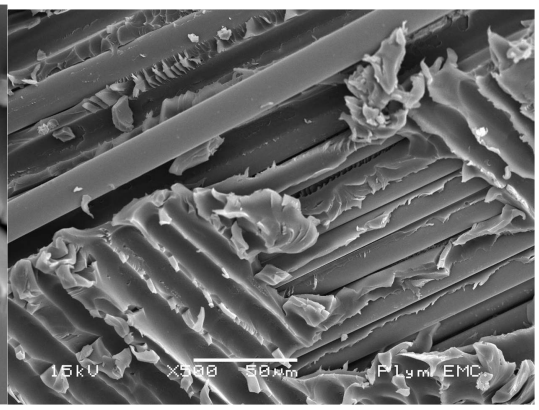


(a) Delamination (from MERL)

(b) Fibre pull-out (from Gleich & Jackson (2002))



(c) Fibre breakage (from University of Plymouth)



(d) Fibre-matrix debonding (from University of Plymouth)

(e) Matrix cracking (from University of Texas at Arlington)

Figure 1.3: Examples of failure in laminated composite

breaking problems share a continuous change in the geometry of the domain under study and their analysis with conventional methods could be a cumbersome and expensive task as well as inaccurate. In fact, for crack propagation problems, FEM is not well suited to treat discontinuities that do not coincide with the original mesh lines. Thus, the most viable strategy for dealing with moving discontinuities in methods based on meshes would be to re-mesh in each step of the evolution so that mesh lines remain coincident with the discontinuities throughout the evolution of the problem.

Also, the analysis of large deformations problems by the FEM, may require the continuous re-meshing of the domain to avoid the breakdown of the calculation due to excessive mesh distortion, which can either end the computation or lead to drastic deterioration of accuracy. In addition, FEM often requires a very fine mesh in problems with high gradients or with local character, which can be again computationally expensive. Also, in order to compare successive stages of the problem, all the meshes need to be stored, requiring more storage memory. Furthermore, for fragmentation problems, like blasts or explosions, it seems natural to choose *particle*-based methods (or node-based) rather than *mesh-based* methods.

In FEM it is very complicated to model the breakage into a large number of fragments as FEM is intrinsically based on continuum mechanics, where elements cannot be broken. The elements thus have to be either totally eroded or stay as a whole piece, but this leads to a misrepresentation of the fracture path. Serious errors can then occur because the nature of the problem is non-linear. To overcome these problems related with the existence of a mesh, a numerical scheme that relies only on nodes would be highly appreciated. These methods are called *meshfree* or *meshless* [1], since they do not need a mesh to construct the *approximation* of the solution of a given differential equation (figures 1.4 and 1.5). According to Liu (2003), "*the minimum requirement of a meshfree method is that a predefined mesh is not necessary, at least in field variable interpolation*". The meaning of this sentence will be clearer in the next chapters. Almost identical definitions can be found in other review papers as Belytschko *et al.* (1996b),

[1] These two terms will be used indifferently in this thesis

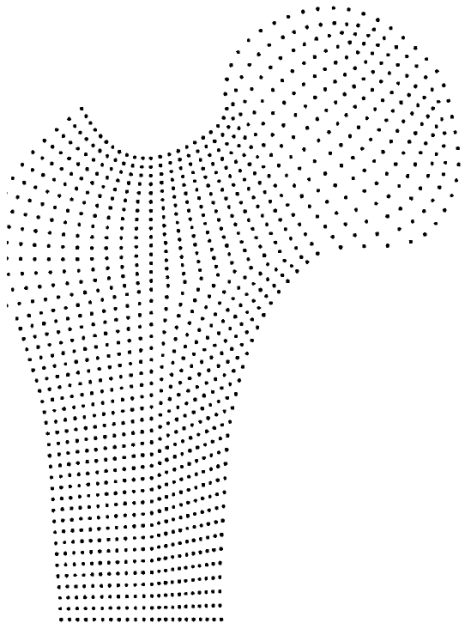


Figure 1.4: Example of Meshfree Method: nodes discretization of a femur (from Doblare *et al.* (2005))

Duarte (1995), Fries & Matthies (2004), Gu (2005) Nguyen *et al.* (2008). More details on classifications and definitions will be provided in section 2.2.

The ideal requirement would be, quoting Liu (2003), "*that no mesh is necessary at all throughout the process of solving the problem of given arbitrary geometry governed by partial differential system equations, subject to all kinds of boundary conditions*".

Another interesting but different point of view can be found in Idelsohn & Oñate (2006). In this paper, even though the authors agree on considering as meshfree a method where the shape functions depend only on the node positions, this definition is not enough to justify the use of a meshless method. In fact, what they consider as discriminating factor is the evaluation of the nodal connectivity. According to them, a meshless method without a fast evaluation of the nodal connectivity is useless. An additional requisite should be therefore added to the definition of meshfree, i.e. that the computational complexity of the nodal connectivity should be bounded in time and linear in number of operations with the number of points in the domain. The problem of connectivity will be discussed

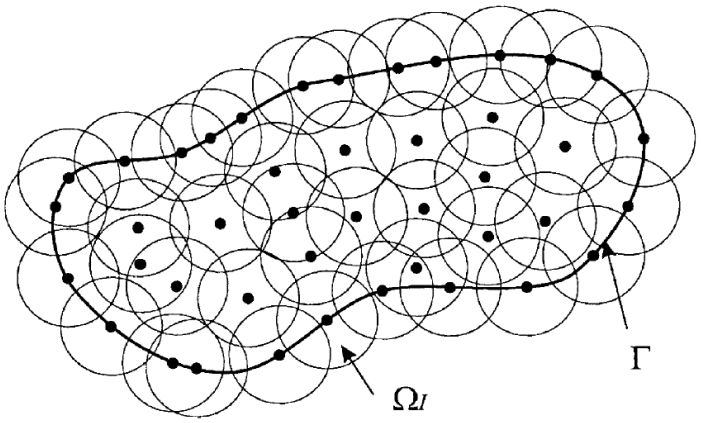


Figure 1.5: Discretization in Meshless Methods (from Doblare *et al.* (2005))

in a later chapter, although connectivity in this thesis will be intended not only between the nodes, but also between nodes and Gaussian points. An algorithm taken from the graph theory, known as *kd-tree*, will be used for this purpose, with the aim of reducing the computational times related to connectivity and to derive a fast method to assemble the matrices of the discretized problem.

1.2 Main differences between Finite Elements and Meshfree

Before proceeding with the definition of the objectives of this thesis, it is necessary to know the major differences (and/or similarities) between FE and MM, to better individuate the weakness and the margin of improvements of the meshfree techniques. Table 1.1, taken from Liu (2003), better resumes the major points.

Point 1 of table 1.1 illustrates the distinguishing difference between the two methods, i.e. the absence of a mesh. This point it is still debatable, since some meshfree methods do require a mesh, but only for *integration purposes*. However, point 4 better clarifies point 1, since the construction of the approximation is *element based* in FE and *point based* in MM. This makes a huge difference in terms of the approximation to the differential equations and the (better) reproducibility properties (and thus accuracy) that MM can achieve compared to FE. The term *reproducibility* will be better explained in section 2.2.

	FEM	**MM**
1 Element mesh	Yes	No
2 Mesh creation and automation	Difficult due to the need for element connectivity	Relatively easy and no connectivity is required
3 Mesh automation and adaptive analysis	Difficult for 3D cases	Can always be done
4 Shape function creation	Element based	Node Based
5 Shape function property	Satisfy Kronecker delta conditions	May or may not satisfy Kronecker delta conditions depending on the method used
6 Discretized system stiffness matrix	Banded and symmetrical	Banded but symmetry depends on the method
7 Imposition of essential boundary condition	Easy and standard	Special methods may be required; it depends on the method
8 Computation speed	Fast	1.1 to 50 times slower compared to the FEM depending on the method used
9 Retrieval of results	Special technique required	Standard routine
10 Accuracy	Accurate compared with FDM	Can be more accurate than FEM
11 Stage of development	Very well developed	Infancy, with many challenging problems
12 Commercial software package availability	Many	Very few and close to none

Table 1.1: Main differences between FE and MM; taken from Liu (2003)

The problem of the reproducing properties of FE is also contained *partially* in point 2 of table 1.1. Having a higher order reproducibility capability usually means that the approximation is more accurate. But, to achieve higher order reproducibility properties, elements might be also of different shapes. A typical example is the choice between triangular elements and quadrilateral elements. Triangular elements with 3 nodes have linear reproducibility while quadrilateral elements with 4 nodes can also reproduce the function xy. While a triangular mesh is relatively easy to construct in an automatic manner, even for complicated shapes, a quadrilateral mesh is usually more complicated to build automatically.

A software able to calculate a *mesh* is called *meshers*. A *mesher* able to mesh arbitrary shapes with quadrilateral elements may not be as robust and may not always work for complex surfaces (or volumes for hexagonal elements). The algorithms behind the meshers are matter of a discipline called *computational geometry* and there exists a vast literature for this topic Preparata & Shamos (1985), De Berg *et al.* (2008). Nevertheless, even if a triangular mesh is *easy* to build, not all the triangular meshes are equal. In fact it is defined a *quality factor* for the meshes, that is an element-wise measure that depends on the ratio between the area of the element and the sum of the squares of the length of the edges of the element Bank (1990). Ideally, an optimal mesh is made only by equilateral triangles. Of course this is not always possible; hence the quality of a mesh must be checked on a case basis. Therefore, a mesh can really affect a FE simulation, and not all the meshes are the same. Obtaining a good mesh can then be difficult, since it is not a mere triangulation of points.

The meshfree methods that require a mesh do not need to submit to these restrictions. In fact, meshes do not have any influence whatsoever on the *approximation* resulting from a MM. Mesh connectivity is a serious problem in mesh-based calculations for cracks. For crack problems, with MM there is no need of costly *re-meshing*. This also applies especially in problems with large deformations or moving discontinuities. In fact, FE mesh distortions (figure 1.6) in large deformation problems can compromise the solution, since most commercial codes normally arrest the algorithm, unless special numerical expedients are performed, like *elements erosion* or the Jacobian sign check. Conversely to FE, MM do not need to define a priori connectivity, neither suffers of mesh alignment

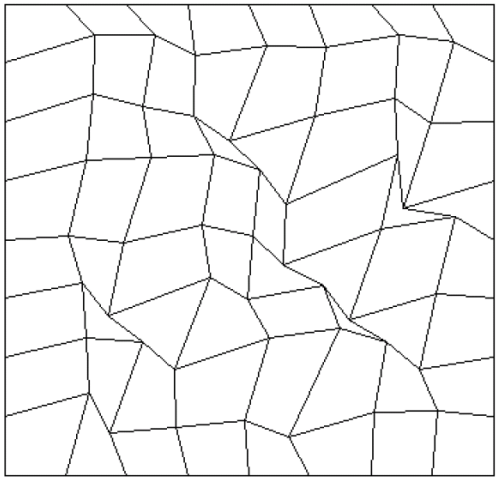


Figure 1.6: Mesh Distortion in FE

sensitivity. However, MM could benefit from the connectivity between the nodes and the Gaussian points. This will be better explained in the next chapters.

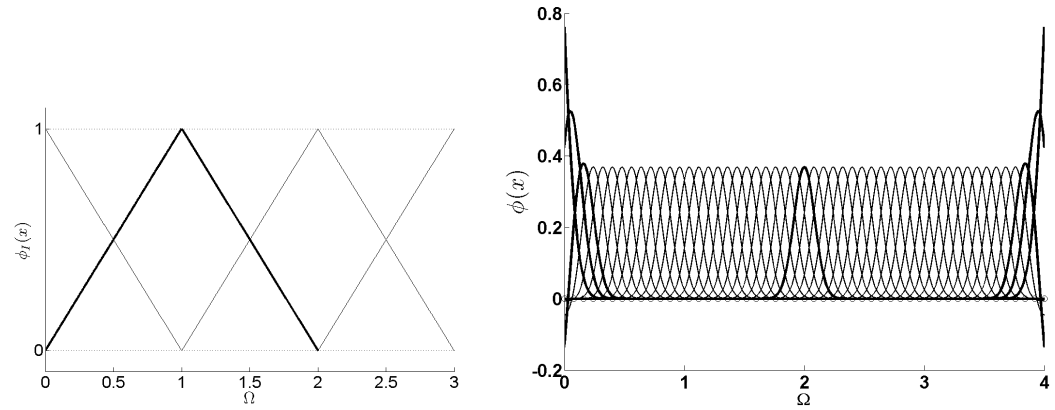
Adaptivity (point 3 of table 1.1) is a very important topic, especially when dealing with high gradient localized zones. In FE it is a difficult task, while in MM it can be done quite easily. A method to efficiently adapt the meshfree shape functions will be presented in the next chapters. In term of *h-adaptivity*, it is comparably simple with MM as only nodes have to be added, and the connectivity is then computed at run-time automatically. In Li & Liu (2004a) is even stated that, contrarily to finite elements, meshfree adaptivity is *infinitely approachable*, i.e. there is no limit on successive *h-refinement*. Also *p-adaptivity* is also conceptually simpler than in mesh-based, since only an additional enrichment (intrinsic or extrinsic) need to be added to the basis functions. This also applies to the inclusion of *a priori* knowledge of the local behaviour of the solution and provides a better framework for multiscale simulations, since enrichments of the finer scales can be incorporated into the coarse scale approximation Li & Liu (2004a).

From a computational point of view, (point 6 1.1), the discretized resulting algebraic system of equations is *banded* and *symmetrical* in the FEM, which means ease of solving when the size of the matrix becomes very large (order of magnitude of thousands). Particularly, FE's stiffness matrices are *banded*, meaning that the majority of the entries of the matrix are zeros, except from those

close to the main diagonal. The number of non-zero entries in the same row close to the diagonal is called *bandwidth*. If the bandwidth is small and constant for all the rows, then these entries can be efficiently stored, since it is necessary to store a fixed number of columns much smaller than the number of rows. Moreover, if the matrix is symmetrical, the number of columns halves. From a numerical point of view, solving linear system of equations with this particular structure is relatively easy and more suitable for large system of equations. For MM, the stiffness matrix could be banded, but the bandwidth could be slightly larger than FE and not constant, although the stiffness matrix is symmetrical, depending on the method used [1]. Nonetheless, the structure is more generally *sparse*, which means that the majority of the entries are non-zero. In this case, only the row and column indexes along with the entry need to be stored. Moreover, there exist methods able to efficiently solve sparse systems of equations Duff *et al.* (1989).

Another major disadvantage of the FEM is the post-processing (point 9 1.1). Re-meshing is not just computationally expensive, but it also degrades the accuracy of the solution. Even in the post-processing of re-meshed solids requires considerable effort, since one need to keep track of all the meshes for example during the time-histories. MM on the contrary have a *post-processing* nature, because shape functions are natively smooth. Any order of continuity can also be achieved adjusting the weight functions and adding more polynomials to the basis, reaching the desired order according to the order of the differential equations. For example, in figure 1.7 are compared MM shape functions and FE linear shape functions. MM shape functions 1.7b are much smoother than the FE ones 1.7a for the same order of the basis. Let us suppose that linear FE shape functions are used to carry out a stress analysis. In stress calculations, the stresses obtained using FEM packages are discontinuous and less accurate, because the simplest shape functions used in FEM are usually linear (figure 1.7a) or however low-order polynomials. Thus, since displacements fields are of the same order of the approximating functions, their derivatives (stress fields) are one order lower. If linear functions are used for example, stress field will be piecewise constant, which could be an unsatisfactory description of the stress distribution. In these cases, stress smoothing techniques must be used, especially for stress recovery

[1] The methods used in this thesis generate symmetric stiffness matrices



(a) Linear FE Shape Functions (b) Same order MM Shape Functions

Figure 1.7: Difference between FE and meshless shape functions

in error estimation. On the other hand, meshfree shape functions 1.7b have smoother derivatives, which enable a quite accurate smooth stress description. In this sense, MM shape functions have a native *post-processing* nature, since they do not need stress smoothing techniques. Also, when smooth trends are not desired, e.g. cracks, discontinuities can be easily handled.

Despite all the attractive characteristics, MM are not medications for all ills. In practice, for a given reasonable accuracy, MM are often considerably more time-consuming than their mesh-based counterparts, because at each sampling point it is often necessary to search for neighbors, solve small systems of equations and perform small matrix-matrix and matrix-vector operations. Furthermore, meshfree shape functions are of a more complex nature than the polynomial-like shape functions of mesh-based methods (figure 1.7). Consequently, the number of integration points for a sufficiently accurate evaluation of the integrals of the weak form is considerably larger in MM than in mesh-based methods. Moreover, the choice of the background mesh containing the integration cells is critical and it easily leads to errors in the patch test Dolbow & Belytschko (1999). According to Dolbow & Belytschko (1999), a *bounding box technique* which takes into account the overlapping supports sensibly reduces the integration errors and can also solve equations with discontinuous parameters. In collocation MM no integration is required; however, this advantage is often compensated by less accu-

racy and more stability problems. In addition, conversely to FE, meshfree shape functions unfortunately do not satisfy (depends on the method used) the Kronecker condition (3.3), (point 5 1.1), thus essential boundary conditions cannot be directly imposed. Essential boundary conditions in MM require opportune treatments (point 7 1.1), like the use of penalty-methods Fernández-Méndez & Huerta (2004), Gavete *et al.* (2000) and Günther & Liu (1998) or Lagrange multipliers Belytschko *et al.* (1996b).

It is evident from the table 1.1, that beside the many advantages of the MM, still their infancy (with the mentioned problems) and the lack of dedicated commercial packages prevent their wide acceptance in the engineering community.

1.3 Objectives

The industrial sector has been relying on FE for decades. Well-developed commercial codes, even for multi-physics simulations, integration with CAD software, not to mention the well-established mathematical work made by researchers in the past years.

It is understandable that industry is often reluctant to open to new methodologies, especially if these new methods cannot offer *better* performances of FE in terms of reliability and speed. This is also the reason why a compromise that put together the benefits of both methods seems to catch on. This compromise is the Extended Finite Element Method (XFEM) Black & Belytschko (1999), Moes *et al.* (1999). The invention of XFEM is subsequent to the invention of MM [1] and, in a philosophical perspective it could be thought as the synthesis between thesis (FEM) and antithesis (meshfree).

The XFEM is essentially FEM with *enrichments*. The meaning of the word *enrichment* will be well explained in the next sections and represents the core of this thesis. For the purpose of this introduction, it suffices to say that these enrichments are able to simulate discontinuities independently from a mesh, even though the method is *mesh-based*.XFEM could be more acceptable for industry because it is essentially a FE which overcomes the limitations due to the presence

[1] The first meshfree paper appeared in 1994 whereas the first paper on XFEM appeared around 1999

of a mesh concerning fracture mechanics problems. A wide literature exists for this topic and review papers can be found in Abdelaziz & Hamouine (2008), Karihaloo & Xiao (2003), Mohammadi (2008). However, XFEM are still *mesh-based* therefore they inherit the other disadvantages of FE, like mesh-distortion, piecewise stress representation for linear elements and so on.

An interesting thread of discussion about the *future of meshless methods* can be found on the website imechanica.org.

The aim of this thesis is NOT to prove that MM are completely alternative to FE, since for many applications FE represents an unbeatable standard, but rather to study their possible applications to a more narrow class of problems, like the failure of composite materials. Particularly it is studied the capability of MM to simulate strong discontinuities as occurs for example in delaminations. More complex failure mechanisms like matrix failure and fibers breakage, originate at smaller length scales (micro-scale) is part of a broader research topic called *multi-scale methods* and will not be covered in this thesis. More reliability and speed is requested to MM or PUM methods to represent a remedy for the disadvantages of FE. A welcome outcome would be the integration of MM in commercial codes or the development of brand new commercial packages.

That brings to the objectives of this thesis. Broadly speaking they are:

1. The production of prototype codes for 2D and 3D simulations, to be further developed for large scale simulations

2. The applications to the simulation of macroscopic failure in composite materials and analysis of advantages and disadvantages deriving from the application of MM

3. The reduction of computational times of MM.

The main outcome of this research has been the production of *object-oriented* codes developed in MATLAB. Even though `Matlab` is not suitable to large scale simulations, it is still quite easy to use, to debug errors and to post-process the results, therefore it is a useful tool to *prototype* a code since one does not have to spend time in re-writing well-established numerical routines, like the solution of linear systems of equation, matrix products and so on. Even if it is believed to be

not as fast as traditional programming languages like Fortran, with appropriate programming tricks and *compilation* of the codes, `Matlab` can be reasonably fast for *medium-size* applications (with number of unknowns order of magnitude of thousand and number of Gaussian point order of hundreds of thousands).

The codes served as basis for the simulation of delamination, proposed in the final chapters. From a geometric point of view, delamination is modeled with an enrichment that exploits the property of *partition of unity* of the meshfree shape functions. The physics of the delamination process will be simulated with a constitutive model called *cohesive zone*, derived from the damage mechanics. The combination of the cohesive law with MM will result in the *cohesive segments* method.

For the third point, the innovations that will be proposed are

1. the employment of a more efficient *node searching* method known as *KD-trees*

2. a faster assembly procedure based on a connectivity map between nodes and Gaussian points

3. a new *matrix-inversion* procedure that also facilitates *hp-adaptivity*

Finally, from a more theoretical point of view, the explicit integration in Reproducing Kernel Methods has been published by the author. More details can be found in appendix A.

1.4 Outline of the thesis

The outline of the thesis is the following: in chapter 2 the literature and the chronological history on meshfree methods will be reviewed, with particular attention to fracture mechanics and composite materials; in chapter 3 the main meshfree methods will be described in detail; subsequently in chapter 4 some improvements to the existing techniques will be proposed, along with results and considerations; chapter 5 will provide a description of the codes developed, while in chapter 6 MM will be applied to the simulation of delamination. Finally, conclusions are presented in chapter 7.

Chapter 2

Literature Review

In this section meshfree methods will be firstly classified and a survey of the existing literature on the topic will be presented. Successively, the review will cover particularly two methods, the Element Free Galerkin (EFG) and the Reproducing Kernel Particle Method (RKPM) . It has been proven that these two approaches lead to identical shape functions Liu *et al.* (1997b), Li & Liu (1996), Jin *et al.* (2001), even if they start from different ideas.

This is a frequent problem in MM. Being a rapidly developing method, confusion and disorder about different methods may arise. In fact in a recent paper Orkisz & Krok (2008) it is reported that there exist about hundred different names of the MM, mostly presenting very similar or even identical ideas. Some of the methods differ only by the names of the authors or by the methods' name, and this certainly leads to confusion.

2.1 Review Papers and Books on Meshfree Methods

A good number of review papers and books on meshfree methods have been published in the recent years. The first review appeared in Duarte (1995) almost at the beginning of the MM era, shortly followed by the more general overview Belytschko *et al.* (1996b). While Duarte (1995) is more focused on the mathematical properties of these methods, Belytschko *et al.* (1996b) is more complete

and more suitable for an engineering audience, since it contains many details on the implementation of MM, especially for applications to crack problems.

The review Shaofan & Liu (2002) instead emphasizes the MM applications for large deformation problems and reviews with particular attention multiscale and particle methods. A subclass of particle methods is the molecular dynamics, widely employed in computational chemistry. This paper is the background of the book Li & Liu (2004b).

The first book on MM is instead Liu (2003) and it is in the author's opinion the most complete work on meshfree methods, since it contains many examples (beams, shells, plates) and comparisons with different methods. A demonstrative software has been also produced from the authors called MFREE2D, although such software does not allow treatment of cracks or enrichments.

Updated reviews on meshfree methods can be found in Fries & Matthies (2004) and Nguyen *et al.* (2008). Fries & Matthies (2004) reports also a classification scheme to better discriminate the different MM, while Nguyen *et al.* (2008) contains also a Matlab *didactic* code.

Special issues in international journals have been dedicated to MM Liu (1996), Chen (2000) and Chen (2004).

2.2 Classification of Meshfree Methods

Several classifications have been proposed, but the debate is currently ongoing.

Liu (2003) proposed a classification based on the point of view that all the meshless methods (MM) so far are not ideal and fail in one of the following categories:

- methods that require **background cells** for the integration of system matrices derived from the weak form over the problem domain. These methods are *not truly mesh free*. These methods are practical in many ways, as the creation of a background mesh is generally feasible and can always be automated using a triangular mesh for two-dimensional (2D) domains and a tetrahedral mesh for three-dimensional (3D) domains. It needs to be stressed out that this mesh is necessary only for integration purposes;

this means that the driving criterion in the generation of such a mesh is the minimization of the integration error rather than the accuracy of the approximation.

- Methods that require **background cells locally** for the integration of system matrices over the problem domain. These methods are said to be *truly mesh free* because creating a local mesh is easier than creating a mesh for the entire problem domain and it can be performed automatically without any predefinition for the local mesh.

- Methods that do not require a mesh at all, but that are less stable and less accurate. Collocation methods and finite difference methods fall into this category. This type of method has a very significant advantage: it is very easy to implement, because no integration is required.

- Particle methods that require a predefinition of particles for their volumes or masses. The algorithm will then carry out the analysis even if the problem domain undergoes extremely large deformation and separation.

Fries & Matthies (2004) instead proposed a classification based on three characterizing features:

- the capability of realizing a Partition of Unity of order n with the intrinsic basis

- the presence of enrichments or more generally extrinsic basis

- the use of the test function

The last one discriminate between *collocation-based* methods or *Galerkin* methods. According to Fries & Matthies (2004), a meshfree method can have one or more of these three features. Moreover, it is also individuated a *grey* area containing hybrid methods. Therefore, the classification suggested in Fries & Matthies (2004) is based on the capability to reproduce polynomials up to a chosen degree with or without an *intrinsic basis*. Moreover, the classification can be also conducted on the choice of *test functions*, leading to local, global weak forms or collocation methods.

2.3 Origins of Meshless Methods

The very first meshless method is *Smoothed Particle Hydrodynamics* (SPH) and it was introduced for the study of astrophysical phenomena without boundaries such as exploding stars and dust clouds and it is due to Lucy (1977). Later Monaghan in Monaghan (1982), Monaghan (1988) and Monaghan (1992) provided a more mathematical basis through the means of kernel estimates. Particularly in the paper Monaghan (1982) is demonstrated that the way a function is represented in particle methods is a special case of interpolation using information from a set of points. Moreover general equations for numerical work can be derived without specifying the details of the interpolation method, showing similarities between finite difference, spectral and particle methods.

Even though SPH was initially conceived for solving astrophysics problem, in Libersky & Petschek (1990) SPH is applied for the first time in solid mechanics and further to study dynamic material response under impact Libersky *et al.* (1993). A recent review on the developments of SPH in impact problems can be found in Vignjevic & Campbell (2009). However, the original SPH version suffered from spurious instabilities, like tensile instability Swegle *et al.* (1995). Nevertheless, such instabilities have been successful treated in Vignjevic *et al.* (2000). Moreover, SPH lacks of consistency properties, especially at the boundaries, where SPH is not able to reproduce even the constant function. For these reasons these methods worked well for unbounded problems like astrophysics, where the domain can be certainly considered as infinite. This is not the case in solid mechanics, where the domains under study are bounded; more importantly, at the boundaries, forces, tractions, displacements are usually applied. Therefore correction terms are needed in order to restore consistency.

Corrections to SPH and related treatment of boundary conditions have been proposed in several papers by the Vignjevic and co-workers, for example in Campbell *et al.* (2000) and Vignjevic *et al.* (2006).

In Rabczuk & Eibl (2003), Dilts (1999), Dilts (2000) and Cueto-Felgueroso *et al.* (2004) SPH is used in a Galerkin weak form using MLS shape functions (*MLSSPH*).

It is important here to recall the concept of consistency because it is crucial for the convergence properties of any numerical scheme. Liu (2003) distinguishes between *reproducibility* and *consistency*. Reproducibility for MM is defined as the ability of the shape functions to *reproduce* exactly all the functions in its basis, whilst consistency is the property of reproducing polynomial functions.

This is due to the fact that if the unknown solution $u(x)$ is expanded in Taylor series around a generic point x, since the approximation reproduces for example polynomials up to degree n, then the approximation is able to reproduce the solution of a differential problem until the order n. It is easy to understand therefore that consistency is a highly desired property for a numerical method. If the basis contains *only* polynomials, then *reproducibility* and *consistency* are equivalent. If the basis contains *only* functions different from polynomials, the approximation has only reproducibility but it is not consistent, hence it may not converge to the solution. If the basis contains both polynomials and *other* functions, the basis is called *enriched* or *enhanced* and it is not only consistent but also able to reproduce these *other* functions. If these *other* functions are for example known solutions of the equations, this means that the approximation is able to reproduce the exact solution of the problem. This concept is the foundation of the enriched methods for crack propagation. In fact, for linear elastic fracture mechanics Anderson (1991), the theoretical solution is singular at the crack tip.

Classical finite elements fail to reproduce accurately the solution. Also, non-enriched meshfree methods are not able to reproduce exactly the solution. Conversely to finite elements, though, meshfree approximation can easily handle the discontinuity while finite elements need elements conforming to the crack. Moreover, using enrichments, it is possible to reproduce singular functions because these singularities can be included in the basis. This is one of the most attractive features of MM.

2.4 Reproducing Kernel Methods and Element Free Galerkin

While SPH and their corrected versions were based on a strong form, other methods were developed in the 1990s, based on a weak form. The first of them is based on a global weak form and it was invented by Belytschko and his coworkers, who exploited the *Moving Least Squares* (MLS) approximation Lancaster & Salkauskas (1981). MLS have been already used to generalize finite elements in the work of Nayroles *et al.* (1992), where *diffuse elements* were proposed. The MLS has its origins in the computer graphics to smooth and interpolate scattered data, thus it seems reasonable to apply this method when only scattered nodes are available as a result of a discretization.

Belytschko *et al.* (1994b) refined and modified the work of Nayroles and called their method *Element Free Galerkin* EFG, which is currently one of the most used meshless methods. This class of methods is consistent, up to *n-th* degree depending on the polynomials used in the basis function, and quite stable although substantially more expensive than SPH.

One year later Liu *et al.* (1995) developed the *Reproducing Kernel Particle Method* (RKPM) which uses wavelets theory, contrarily to MLS. Surprisingly, the imposition of the reproducibility for the polynomials led to shape functions almost identical to MLS.

2.4.1 Reproducing Kernel Method

Reproducing Kernel Method (RKM) has its origins in wavelets theory Liu *et al.* (1996) and its discrete counterpart has given birth to Reproducing Kernel Particle Method (RKPM) which is one of the emerging classes of methods called meshless methods. Moreover, RKPM shares similarities with *moving least squares* (MLS) method and RKM is ultimately more a category within these methods can be classified Liu *et al.* (1997b), even though MLS has its origins in data fitting. Indeed, sometimes RKM is referred to as a *corrected SPH*.

In standard Smoothed Particle Hydrodynamics (SPH) method Gingold & Monaghan (1977), Monaghan (1992), the function is simply approximated by an

its convolution with a *kernel* function, which satisfies a set of properties. One of them is that the kernel is a compact support function with dilatation parameter ρ. The approximation is then filtered conferring a smoother behaviour to the approximation. SPH scheme though, suffers of inconsistencies at the boundaries, precisely at a distance equal to ρ Liu *et al.* (1997a). SPH is not able to reproduce even the constant function, i.e. it does not realize a partition of unity Belytschko *et al.* (1996b). This is also the reason SPH is more suitable for unbounded domains.

In order to restore this condition, a corrected kernel must be used. It can be demonstrated that using a modified kernel Liu *et al.* (1996) by the means of a *moments matrix*, the reproducibility condition up to order n is restored. This *moments matrix* has entries that are essentially the kernel estimates of polynomial functions up to degree $2n$. The resulting approximation is known as Reproducing Kernel Method (RKM).

Although this theory is mathematically correct, its implementation might pose problems since it is necessary to evaluate integrals. Thus in real computations a discretization is often necessary. This discretization is based on *particles* associated with a measure of the domain and the integrals are replaced with summations. However, an explicit computation of the integrals involved in RKM without discrete summations has been proposed in Barbieri & Meo (2009b). A description of this method will be proposed in the next chapters.

While in RKM the moments matrix depends only on the domain (actually, only on the domain's boundaries Barbieri & Meo (2009b)), in RKPM, because of the discrete sums, it depends on the particles distribution. RKPM therefore associates the advantage of being meshless with a higher accuracy, keeping the multi-resolution property assured by the dilatation parameter of the kernel.

In Liu *et al.* (1996) and Liu *et al.* (1997a) the wavelets character of RKPM allowed to generalize the method to multiple length scales. In Liu *et al.* (1996) is proposed an approach to unify all the reproducing kernel methods under one large family and an extension to include time and spatial shifting. In the latter Liu *et al.* (1997a) the approach is particularized for its particle version RKPM. The Fourier analysis is used to address error estimation and convergence properties.

2.4.2 Element Free Galerkin

In parallel, Belytschko *et al.* (1994b) applied another meshless approximation scheme called moving least squares (MLS) Lancaster & Salkauskas (1981) to a Galerkin approach creating the *Element Free Galerkin* (EFG) method. In MLS a function is approximated by a weighted least squares procedure where the weighting function is *moving* in the sense that it depends on the evaluation point. The weighting function plays the same role of the kernel function in RKM, since it is a smooth compact support function where the size of the support is ρ. The support is centered on the evaluation point, around which fall a certain number of nodes. The result is that the function is not interpolated, but *approximated* at the nodes (similarly to the particles).

Thus, the resulting minimization of the sum of the squares of the error leads to solving a linear system of equations in each point of interest. The matrix of this system is called *moment matrix* and in fact it is the same of the RKPM. The similarities between the two methods became more evident in the *Moving least-square reproducing kernel methods* (MLSRKM), where a general framework is introduced Liu *et al.* (1997b), Li & Liu (1996). From MLSRKM , both RKPM and MLS can be derived, where the sum of the square errors can be interpreted in a continuous way if an inner product based on integrals is considered. These integrals are the same introduced in the *moment* matrix.

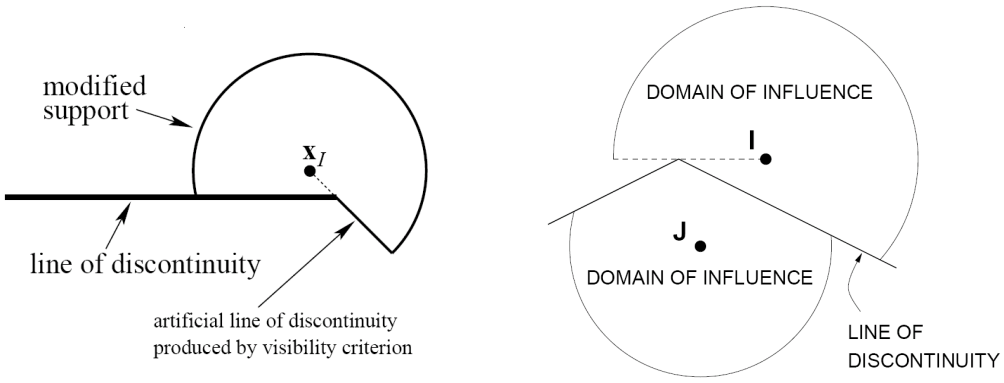
Applications of both RKPM and EFG have been quite successful in the recent years, especially in problems with discontinuities and singularities and a large number of papers can be found which dealt with these questions. The leap in these methods has been provided by the works of Melenk & Babuška (1996) and Duarte & Oden (1996) where it is recognized that MLS approximation is a particular *partition of unity.*

The essence of the partition of unity method relies on the fact that whatever partition of unity can be multiplied with any independent basis to form a new and better basis. Sometimes the choice of an independent basis can be based on users' prior knowledge and experience about the problem. This led to the creation of the enrichments for MLS (or RKPM) shape functions, which have been wisely and widely exploited for fracture mechanics problem.

2.5 Applications to Fracture Mechanics

There is a vast amount of literature about applications of MLS or RKPM in fracture mechanics, most of them by the Northwestern University group of Belytschko and W.K.Liu. Probably Belytschko *et al.* (1994a) is the first application of EFG to crack problem contemporary to the publication of the classic paper Belytschko *et al.* (1994b).

In Belytschko *et al.* (1994a) a *visibility* criterion is applied to model discontinuities in the domain. This criterion is the very first one that has been applied and it was subsequently modified to give more accurate representations of a crack tip.



(a) (b) failure for non-convex domains

Figure 2.1: Visibility criterion (from Belytschko *et al.* (1994a))

Lagrange multipliers can be used to enforce essential boundary conditions, even if penalty method is mentioned as well. A crack propagation criterion based on the stress intensity factors (SIF) is used, and the SIFs are calculated according to the famous *J-integral* Rice (1968) opportunely converted in a domain integrals. All the integrals are evaluated in this paper with the mean of Gauss quadrature with special mention to the problems that can arise. Also, the distribution of nodes is a circular arrangement around the crack tip which can effectively capture the stress singularity and the SIFs. Moreover it is possible to move a dense

arrangement of nodes around a crack tip without re-meshing, which is the greatest advantage over FE methods.

In Belytschko *et al.* (1995a) the classic problem of a plate with circular hole under remote tension loading is presented. (figure 2.2)

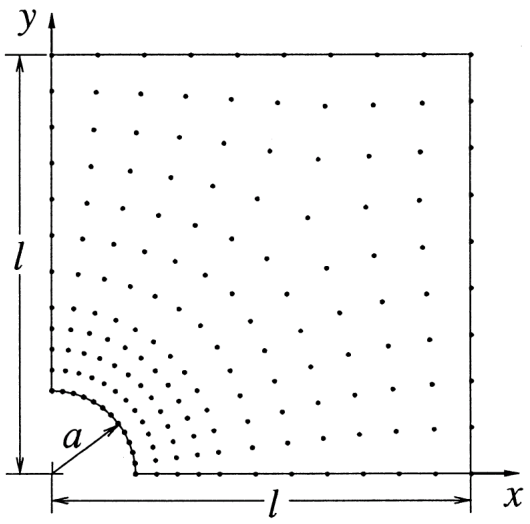


Figure 2.2: Plate with circular hole under remote tension (from Belytschko *et al.* (1995a))

On the outer boundaries the exact solution in terms of stresses is applied and due to symmetry, only a quadrant of the problem is considered. Results show that when concentrating the nodes distribution nearby the hole, the calculated stresses are identical to the analytical ones. Regarding the edge crack problem, SIFs are calculated for different ratio of width and length showing good comparison with known solutions. Instead of visibility criterion, each node lying on the crack line is split in two nodes, separated from a very small distance. It is shown that to obtain greater accuracy, not only the number of nodes needs to be increased, but also they need to be arranged along circular line around the crack tip (figure 2.3). In fact, regular arrangement even with more nodes gives less accurate results than the circular arrangement.

The two cases are then combined in a problem with cracks emanating from a circular hole in order to study crack propagation. A criterion based on SIF

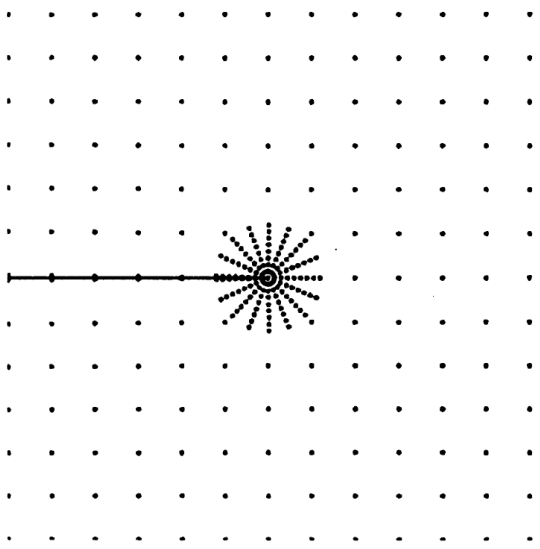


Figure 2.3: Typical Nodes Arrangement for edge crack problem (picture taken from Belytschko *et al.* (1996a))

and angle θ for the direction of the crack growth is used to predict the propagation path, derived from the Paris-Erdogan law for fatigue crack growth Paris & Erdogan (1963).

Same approach with identical procedures is proposed in Belytschko *et al.* (1995b), with the addition of a simulation of cracks in a dynamic field, i.e. how a crack propagates in a moving body. Integration in time is carried out with a β-Newmark integrator and solutions for dynamic stress factors are in agreement with the analytical solution by Freund (1990). Another important work in dynamic fracture mechanics with similar results is Belytschko & Tabbara (1996).

In Belytschko *et al.* (1996a) a new procedure to treat discontinuities is firstly introduced, which overcome the disadvantages in the visibility criterion Belytschko *et al.* (1994a). In fact visibility criterion fails with non-convex boundaries (figure 2.1b). This new procedure is the *diffraction* criterion (figure 2.4) and it is based on the relative distance among each sampling point $\mathbf{x}$, the discretization nodes $\mathbf{x}_I$ and the crack tip $\mathbf{x}_c$. With this procedure, nodes hidden by a non-convex boundary are considered, leading to new weight functions capable of handling the presence of a line of discontinuity. Also, in the same paper, a procedure to accelerate the computation of shape function is introduced for the first time.

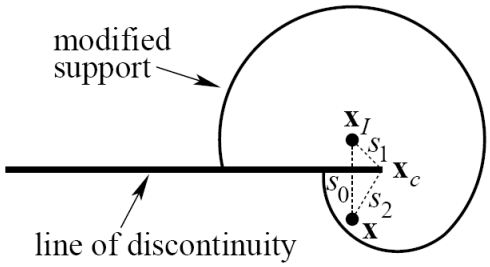


Figure 2.4: Diffraction criterion; figure taken from Belytschko *et al.* (1996a)

Since shape functions are the results of small system of equations, *LU factorization* can be used and the factorized matrices $\mathbf{L}$ and $\mathbf{U}$ stored for subsequent usage in the derivatives calculation. This indeed permits to save computational costs.

One of the most important steps forward in MM is surely the work of Duarte & Oden (1996), where the *Partition of Unity* method (PUM) is introduced. Basically, PU functions are constructed starting from a set of N scattered nodes $\mathbf{x}_I$ in a domain Ω. Each nodes is the center of a sub-domain (or *clouds*) Ω_I. On each sub-domain, a function $\phi_I(\mathbf{x})$ is defined, which is non-zero outside Ω_I (figure 1.5). Differently from FE, these sub-domains overlap, therefore the condition of the partition of unity (i.e. the sum of all these function on Ω is equal to 1) is not automatically guaranteed, but they have to be constructed accordingly.

Apart from classic FE functions, another class of functions that easily realize PU is *Shepard's function* where simply each function is normalized to the sum of all $\phi_I(\mathbf{x})$. From the consistency point of view introduced in subsection 2.3, this class lacks of accuracy, since they can only reproduce exactly constant functions. Their advantage is the ease with these functions can be calculated, since conversely to other MM, they do not need the resolution of a system of linear equations. The importance of this work is that, even starting from Shepard's function with low accuracy, more accurate shape functions can be constructed simply by multiplying Shepard's function with another basis function, which contains *enhanced* or *enriched* functions. These functions could be either polynomials or

functions similar to the known solution of the differential equation, like in Melenk & Babuška (1996) for the Helmholtz equation in one dimension.

This method is also called *extrinsic enrichment* since the augmented basis is external to the basis used to construct $\phi_I(\mathbf{x})$. The greatest drawback of these methods is that more unknowns are introduced. If N is the number of nodes and k the number of enhanced functions, there are $N \times k$ more unknowns than the usual N. Since usually in real computations N could be a very large number, this method can sensibly increase the computational cost. Therefore, usually only few nodes are enriched; for example in edge crack problems, the nodes close to the crack tip are enhanced. The contribution of Duarte & Oden (1996) is that, since MLS shape functions are specific instance of PU (they are actually PU of order $n \geq 1$), they can replace Shepard's functions.

The advantage is that *hp*-adaptivity is much more facilitated. In general adaptivity is a method to assess the quality of the results (based on error estimation theory) and modify the mesh during the solution aiming to achieve approximate solution within some bounds from the exact solution of the continuum problem. Precisely:

- h-adaptivity in FE is when elements are refined (or unrefined). In MM this is done simply adding nodes (figure 2.5)
- p-adaptivity when the order of polynomial basis functions is changed
- *hp*-adaptivity is a combination of the above

Fleming *et al.* (1997) presented for the first time a different way of enrich basis functions, called *intrinsic* enrichment. Because of the reproducibility property of the MLS, enhanced function can be added to the basis. In Fleming *et al.* (1997) the two methods of extrinsic and intrinsic enrichment are compared. The augmented functions are the well known asymptotic solution of the crack tip in linear elastic fracture mechanics, whose stresses are singular at the end of the crack line. Therefore functions whose derivatives are singular need to be considered, for example the square root of the distance between the crack tip and a generic point. In contrast to the extrinsic method, the intrinsic one does not require additional unknowns. However, since the size of the basis is increased,

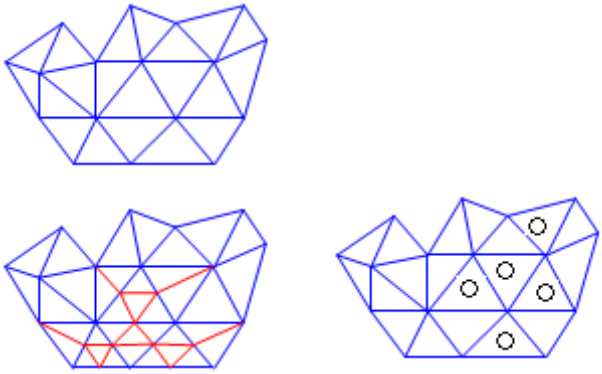


Figure 2.5: h-adaptivity in FE (left) and meshfree (right)

more computational effort is needed to invert the moment matrix. Moreover, for multiple cracks, additional terms are required for each crack. Enriched basis can lead to a moment matrix which exhibits some ill-conditioning. LU factorization is used to alleviate this ill-conditioning. In Fleming *et al.* (1997) is also introduced a smoothing factor to the diffraction method and also a mapping for kinked crack is proposed.

The benefit of the introduction of the enrichment is quite evident. Non-enriched functions cannot reproduce the exact stress field, while enriched methods show perfect agreement with the analytical solution. Also, oscillations are eliminated without extra refining at the crack tip. Convergence considerations about discontinuous approximations introduced by these enrichments are presented in Krysl & Belytschko (1997). Krysl & Belytschko (1997) pointed out that much of the mathematics for FE is applicable to EFG.

Therefore EFG method satisfies all the requirements under which convergence estimates based on interpolation theory apply to conforming finite element method. Consequently it can be expected that this FEM convergence theory will apply also to the EFG method. It is shown that the enrichment greatly increases the absolute accuracy. Accuracy means that the errors are small compared to the exact solution and thus the code provides results *close* to the *truth*. When the solution is unknown, the error is *estimated*.

In fact, for regular EFG (linear basis), the rate of convergence (which means the increase in accuracy due to uniform nodal refinement) for problems with a

singularity is controlled by the order of the singularity for a polynomial basis of any order. For linear basis, the theoretical rate of convergence is therefore 1/2 (because the exact solution goes with the square root of the distance from the crack tip). For the enriched approximations the exact solution is contained in the trial function and according to the theory of minimum potential energy there should be no errors due to approximation. In this case, the errors arise because of quadrature error and discretization of the essential boundary conditions.

The early works with EFG dealt with two-dimensional structures until 1997 when the first applications to three-dimensional domain started to appear. In Sukumar *et al.* (1997) 3D EFG is generalized in a straightforward manner. The ease stands in the fact that, since no elements are required, one simply needs to add function to the basis functions like z or z^2. Weight functions are *tensor products* of one-dimensional weight functions, whereas implicit enriched functions are used to achieve higher accuracy for the crack representation. An extended diffraction criterion is used to create shape functions which reproduce the discontinuity line introduced by the edge stationary crack. While computational technology for stationary cracks is well established, for 3D domains the main issue is the geometrical representation of arbitrary evolving cracks.

MM appears to be one the most promising method to handle even three dimensional cracks. In Krysl & Belytschko (1999) FE are coupled to EFG in the sub-domain where the crack exists. Representation of the crack surface is made by a triangular mesh and segments are used for the crack front. Since kinks in the crack might be present, a smooth representation is not preferred.

A quite complex algorithm called *node inclusion* is presented to create discontinuous shape functions. The crack propagation algorithm is based on SIFs. The addition of triangular elements is made with advance vectors. Recalculation of shape functions is made accordingly. The method is applied successfully to simulation of mixed-mode growth of center-through crack in a finite plate, mode-I surface-breaking penny-shaped crack in a cube, penny-shaped crack growing under general mixed-mode conditions and torsion-tension rectangular bar with center through crack. Comparisons are made with respect to classic methods like FE, showing that complex re-meshing is avoided.

In the comprehensive paper Belytschko & Fleming (1999) the whole EFG method is reviewed. Classic test cases are replicated such as infinite plate with a hole, near-tip crack, plate with a hole and two cracks and compression-loaded cracks. Beside the already known results, an innovation is introduced, precisely the contact between two bodies or self-contact as in cracks problem. The contact is implemented with the same penalty method used to impose essential boundary conditions.

In Ventura *et al.* (2002) a new *vector level set* method for modeling propagating cracks in EFG is presented. With this approach only nodal data are used to describe the crack and no geometrical entity is introduced for the crack trajectory. The *vector level set* basically consists in a local extrinsic enrichment when one of the enhanced functions is a *sign* function of the distance between the *enriched* node and the crack line. Only the nodes whose support is cut by the crack line have this type of enrichment. The nodes close to the crack tip have as enhanced functions the linear elastic fracture mechanics solution for an infinite plate. The update of the vector set is made through an advance vector of the crack tip.

2.6 Applications to Nonlinear Mechanics

As anticipated in section 1.1, MM are also employed in problems with large deformations.

A large deformations in FE easily causes element distortion, resulting in a lack of accuracy. In Chen *et al.* (1996) there is the first application of RKPM to large deformation in hyper-elastic (figure 2.7) and elasto-plastic materials. The advantage is that the compact support of a RKPM material shape function covers the same set of particles during the deformation and hence there is no tension instability. This is achieved with the introduction of a material (i.e. the initial reference configuration) kernel function that deforms with the material, rather than using a kernel with a fixed support. In fact fixed supports could lead to instabilities under large deformations. The usage of material shape functions avoids the re-adjustment of kernel function hence without creating an additional burden.

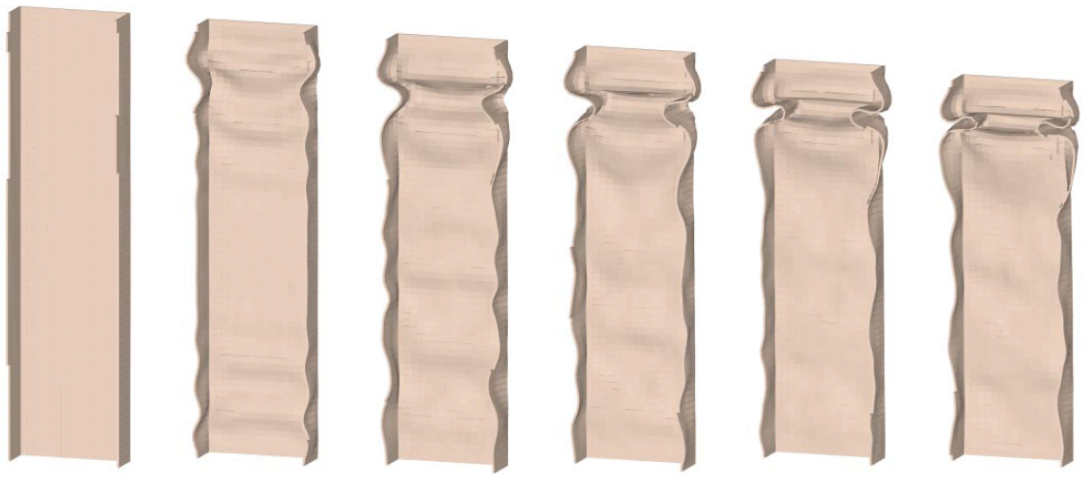


Figure 2.6: Example of Nonlinear Analysis: box-beam under impact; picture taken from Chen *et al.* (1996)

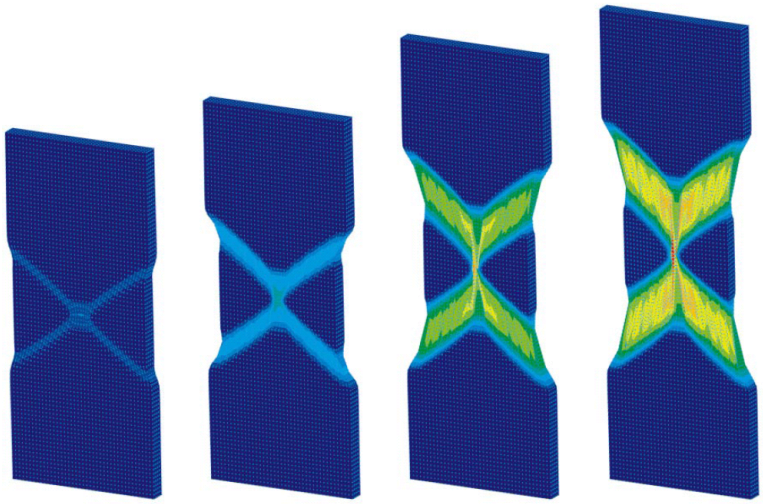


Figure 2.7: Example of Large Deformation Analysis: hyperelastic material tensile test; picture taken from Chen *et al.* (1996)

The same approach is successfully used in Chen *et al.* (1997) to model the behaviour of rubber in test cases like uni-axial tension-compression of a plane-strain rubber unit, shear tests, inflation of an infinitely long rubber tube, disk inflation and the analysis of a silent block rubber bushing assembly.

In Sukky Jun (1998) several issues regarding fast and exact algorithms for large deformation problems have been addressed and it is suggested an algorithm for the simulation of large deformation problems. The method is then applied to a hyper-elastic material and also to a highly non-linear large deformation problem and complex phenomenon as underwater bubble dynamics, to test the goodness of the method.

In Liu & Jun (1998) the *multi-resolution* properties of RKPM Liu *et al.* (1996), Liu *et al.* (1997a) are exploited to detect the different scales of the response of a largely deformed material.In fact, due to this multi-resolution property, the response of a complex mechanical system can be separated into different scales and each scale is investigated separately. Therefore the non-linear Cauchy Green tensor is decomposed in a high scale and a low scale component, where the high-scale component of the solution is useful as a criterion for the procedure of adaptivity.

In Ponthot & Belytschko (1998) an *Arbitrary Lagrangian Eulerian* (ALE) formulation is applied to EFG. ALE is particularly used in fluid-structure interaction problems and a combination of the two classic points of view in continuum mechanics, i.e. Lagrangian and Eulerian description. The reason of using such a combined formulation stays in the fact that neither the Lagrangian nor the Eulerian descriptions are well suited because of their well-known intrinsic limitations. Essentially, the Eulerian description can deal with very large material distortions but have limited spatial resolution to track a moving boundary. Conversely the Lagrangian description can precisely track a moving boundary or interface but is limited by mesh distortions. In fact, the Eulerian and Lagrangian formulations viewpoints have rather complementary virtues. ALE has the possibility to automatically retain an evolving high density of nodes where it is required and in Ponthot & Belytschko (1998) this ability is joined with the ease of EFG in treating discontinuities.

Other interesting applications of RKPM regard simulations of large deformation of thin shell structures Li *et al.* (2000b). These structures are simulated

as 3D continuum, differently from shell elements in FE. Results are quite surprising, showing that RKPM can easily simulate large deformation problems of both elasto-plastic and hyper-elastic thin shell structures like cylindrical shells and conic shells.

Other interesting non-linear applications include shear band formations Li *et al.* (2000a), Liew *et al.* (2002a), plasticity due to impacts Eskandarian *et al.* (2003), Chen *et al.* (2005) and Beissel *et al.* (2006), non-linear fracture mechanics Rao & Rahman (2004) and non-linear analysis of plates and laminates Belinha & Dinis (2007).

2.7 Other Meshfree Methods

As mentioned in section 2.2 there are many different MM, most of them presenting very similar or even identical ideas.

A probably non-comprehensive list of MM is Finite Point Method Onate *et al.* (1996) , the Meshless Local Petrov-Galerkin (MLPG) Atluri & Zhu (1998), Local Boundary Integral Equation (LBIE) Zhu *et al.* (1998), Natural Element Method (NEM) Sukumar *et al.* (1998), Meshless Finite Element Method (MFEM) Idelsohn *et al.* (2003), Finite sphere De & Bathe (2000) and hp-clouds Duarte & Oden (1996).

2.8 Shortcomings, Recent Improvements and Trends

Despite their attractive features, MM still have some drawbacks. The most problematic is the solution of small systems of equation at each point of evaluation, since it really slows down the computational run-time.

Recently efforts have been made in this sense, with the work from Fasshauer (2002) Fasshauer & Zhang (2003). The final goal of these works would be an explicit representation of shape functions, avoiding so the burden of the numerical solution of a linear system of equations. In fact, beside the computational costs, another issue to take into account when dealing with numerical solution of equation is the *conditioning* of the matrix. As for Lagrangian interpolation,

ill-conditioning of the moment matrix may appear whenever polynomials are included in the basis function. One of the most effective remedies is an appropriate scaling and translating of these polynomials.

An important work in this sense is made in Breitkopf *et al.* (2000) and Zhou *et al.* (2005) where the moments matrix is evaluated symbolically, providing explicit representation of the shape functions. It is showed that computation of shape functions in RKM is considerably speeded up. It has been verified by the author that when the moment matrix exceeds the size 5×5, symbolic tools cannot provide an explicit and stable expression for the entries of the inverse. This aspect has been investigated by this thesis' author and an alternative method, based on the inversion of partitioned matrix, will be proposed in the next chapters.

Another issue is related with the quadrature, since meshfree shape functions are usually not integrable explicitly. Therefore a background mesh is necessary to compute the integrals involved in a weak formulation. This is the reason why some MM are called *not truly meshfree*. Duflot & Nguyen-Dang (2002) proposed a quadrature with MLS Shepard Functions which realize a PU. The integration is made on overlapping supports, as also proposed in Dolbow & Belytschko (1999). The difference stays that no background cells are required, making the method *truly* meshfree. The method is both simple and smart because the integration scheme is related to the set of nodes used for the approximation. So, the quadrature points concentrate automatically where the node density is high. Also, it is particularly useful in adaptive computations when an increase in the number of nodes in a given area will involve an increase in the number of quadrature points in this area.

In Khosravifard & Hematiyan (2009) a rather different approach named Cartesian Transformation Method (CTM) is proposed as an alternative to the background regular grid integration. CTM makes use of the Gauss' theorem to transform domain integrals in boundary integrals. The resulting scheme is basically an integration method based on regular grid arrangement with the difference that even complicated boundaries are calculated more accurately. According to the definition proposed in section 2.2, Khosravifard & Hematiyan (2009) CTM is a *truly* meshfree method. However, CTM employs Gaussian points, which can easily outnumber the nodes. This means that the construction of the shape functions

requires evaluations for a large number of points. The ideal requirements would be that the shape functions are evaluated *only on the nodes*, without resorting to Gaussian integration.

This type of integration is called *nodal integration* Beissel & Belytschko (1996). The advantage of fewer evaluation points is on the other hand paid with less accuracy and instabilities, since usually derivatives of the shape functions are integrated in weak forms, and derivatives in nodes are usually zero. Therefore stabilizing terms must be introduced. Different techniques have been proposed to avoid this inconvenience, for example stabilized conforming nodal integration (SCNI) Chen *et al.* (2001) and stabilized nodal averaging Puso *et al.* (2008), where the calculation of the derivatives is not necessary, with undoubted alleviation of the computational burden. Nevertheless these techniques still do not provide the same accuracy of the Gaussian element integration, but further efforts and developments of nodal integration techniques could really lead to a breakthrough in meshfree methods.

Recent improvements have also been achieved in the way the discontinuities are approximated. Duflot & Nguyen-Dang (2004a) Duflot (2006) suggested *enriched weight functions* rather than basis enrichment and more recently Muravin & Turkel (2006b) Muravin & Turkel (2006a) proposed the *spiral weight* criterion alternatively to the diffraction criterion. Spiral weight appears to solve issues related with the diffraction criterion, for example it does not properly capture the stress singularity at the crack tip when a linear basis is used.

Recently two papers dealt with the future trend of MM. Idelsohn & Oñate (2006) discussed the choice between mesh method and a meshless method for the solution of a partial differential equation by a numerical method. In this very interesting paper the author poses some tricky questions as "'*why are so many people trying to use meshless methods?*"' or "'*is better to solve a problem with or without mesh?*"'.

This paper discusses this issue to show that in most of the applications, it is not the right question, whether using a mesh or not. In fact, for Idelsohn & Oñate (2006), if the problem needs a permanent update of the nodal connectivity, the most important requirement is to ask for an algorithm giving a linear relation between the number of nodes and the number of operations to generate

the connectivity. When the problem to be solved needs a permanently update of the nodal connectivity, if the algorithm is based on a mesh or a meshless method is, in general, irrelevant. Both mesh based methods or meshless methods may give the correct answer provided the algorithm used for the evaluation of the node connectivity is bounded in computing time and is linear with the number of nodes. According to Idelsohn & Oñate (2006), this is crucial to decide the appropriate method to use.

In Nguyen *et al.* (2008) is contained the most updated review on MM. In this paper, the problem is posed in terms of competitiveness of meshfree methods against other emerging method such as Extended Finite Elements (XFEM). In fact both methods seem to have advantages and disadvantages, for example the computational costs in MM. XFEM are essentially enriched FEM in presence of cracks, using the PU property expressed in Duarte & Oden (1996). XFEM, as FEM cannot deal with distorted meshes, which would decrease its direct applicability to problems involving high mesh distortion. Moreover, another item for future research concerns the *mathematical* theory of meshfree methods. A unified theory of the approximation properties and stability of meshfree methods has attracted recent interests, and still a potential unified theory appears to be missing.

The influence of the shape and size of the domains of influence on accuracy and stability in MM based on local/global weak forms or collocation methods remains unclear.

Chapter 3

Description of the Method

In the previous chapter the chronological development of the main meshfree methods has been presented. In this chapter the methods will be illustrated more in detail, with particular emphasis on Reproducing Kernel Particle Method. The chapter starts with some definitions and proceeds with describing the approximations in SPH and Reproducing Kernels. Subsequently, different methods for the treatment of discontinuities will be described. Finally, the weak form and the equations of equilibrium are presented in the last section.

3.1 Definitions

In this section are briefly recalled some concepts which are the basis of the MM. Throughout this thesis the function to be approximated will be indicated as $u(\mathbf{x})$ where $\mathbf{x} \in \Omega$ where Ω is the domain under consideration. The approximation will be indicated with a superscript h, as $u^h(\mathbf{x})$ and the following holds

$$u^h(\mathbf{x}) = \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I \tag{3.1}$$

where $\phi_I : \Omega \to \mathbb{R}$ are the *shape functions* and U_I is the *i-th* nodal value located at the position $\mathbf{x}_I$ where $I = 1, \ldots, N$ where N is the total number of nodes. In MM it must be pointed out that most of the methods do not interpolate prescribed values $\mathbf{U}_I$ at the nodes $\mathbf{x}_I$

$$u^h(\mathbf{x}_I) \neq U_I. \tag{3.2}$$

For this reason, the term *approximation* is preferred to the term *interpolation*. Because of equation (3.2), shape functions do not satisfy the Kronecker condition

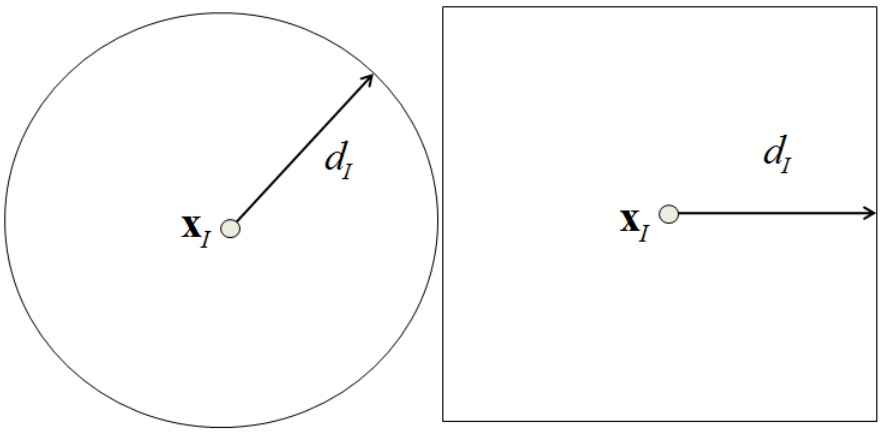
$$\phi_I(\mathbf{x}_J) \neq \delta_{ij} \qquad \forall I, J = 1 \dots N. \tag{3.3}$$

This will affect the imposition of the essential boundary conditions, as it will be shown later.

3.1.1 Kernel or Weight Function

The shape functions ϕ_I are derived from the *kernel* functions, or *window* or *weighting* functions, which are indicated by $w_I : \Omega \to \mathbb{R}$

The kernels have compact support, which means that they are zero outside and on the boundary of a ball Ω_I centered in $\mathbf{x}_I$ (figure 3.1a). The radius of this support is given by a parameter called *dilatation parameter* or *smoothing length* which can be indicated as ρ_I or d_I or a_I to avoid confusion with the mass density. According to the norm considered, the shape of the support may vary, for example it could be a circle but also a rectangle (figure 3.1b).



(a) Circular Support (b) Rectangular Support

Figure 3.1: Examples of compact support

The *dilatation parameter* is crucial for the accuracy, the stability of the algorithm and plays the role of the element size in the FE, although its effects on all

these aspects have not been rigorously demonstrated yet. Moreover, when variable dilatation parameters are used, particular care must be taken in the choice of these quantities. As a rule of thumb, with a background element mesh, it is practical to choose

$$\rho_I = \alpha \max_e l_{eI} \tag{3.4}$$

where α[1] is a scaling factor depending on the degree of the polynomial basis and l_{eI} is the length of the e^{th} edge of the all the elements having $\mathbf{x}_I$ as a node.

When the nodes are distributed non-uniformly, according to (3.4) a variable dilatation parameter is assigned to each particle. As a result, a non-uniform distribution of ρ_I is obtained. Whenever a significant change in node density exists, (e.g. to capture stress concentration, or in adaptivity), the distribution of smoothing lengths over the domain will be therefore non-uniform (high $\nabla\rho$). As a drawback, the resulting approximation (regardless the differential equation) is poor, since it is known that MLS *approximants* Rabczuk & Belytschko (2005) degrade in the presence of sudden change in the support size. A remedy is presented in Rabczuk & Belytschko (2005), it could be, for example, based on averaging the smoothing lengths. Nonetheless, the role of the dilatation parameters is still not well understood and it represents an obscure area in MLS and RKPM, which requires further mathematical work. A deeper look in the field of the approximation theory will hopefully clarify the role of the smoothing lengths and provide a more rigorous rule for their choice.

The fact of being compact basically guarantees the sparsity and the *bandness* of the stiffness matrix in a Galerkin formulation, which is particularly useful for the computer algorithms of matrix inversion. The final characteristic of weight functions is its functional form Fries & Matthies (2004).

Naming

$$s = \frac{\|\mathbf{x} - \mathbf{x}_I\|}{a} \tag{3.5}$$

[1]To avoid singularities of the moment matrix, α should be at least $2 - 2.2$ for quadratic basis in two dimensions, at least 2.4 for cubic polynomials and so on

the properties of kernel functions are

$$w(s) \geq 0 \quad s \in [0,1], \tag{3.6a}$$

$$w(1) = 0, \tag{3.6b}$$

$$\int_0^1 w(s)\mathrm{d}s = \frac{1}{2}, \tag{3.6c}$$

$$\lim_{a \to 0} w(s) = \delta(s), \tag{3.6d}$$

$$w(s) \quad \text{is monotonically decreasing with} \quad s. \tag{3.6e}$$

Properties (3.6a) and (3.6b) guarantee that the kernel has compact support, while (3.6c) is a normalization property. Property (3.6d) assures that at the limit the kernel becomes a *Dirac function*. Some functional expressions of the kernel could be for example the *3rd order spline*

$$w(s) = \begin{cases} \frac{2}{3} - 4s^2 + 4s^3 & 0 \leq s \leq \frac{1}{2} \\ \frac{4}{3} - 4s + 4s^2 - \frac{4}{3}s^3 & \frac{1}{2} < s \leq 1 \\ 0 & s > 1 \end{cases} \tag{3.7}$$

which is $C^2(\Omega)$[1] or more generally the *2k-th order spline* (figure 3.2)

$$w(s) = \begin{cases} (1 - s^2)^k & 0 \leq s \leq 1 \\ 0 & s > 1 \end{cases} \tag{3.8}$$

which is $C^{k-1}(\Omega)$ with $k > 1$.

The order of continuity of a kernel function is important because it influences the order of continuity of the shape functions. First partial derivatives of the kernel with respect to the variable x_k can be evaluated using chain rule

$$\frac{\partial w}{\partial x_k} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial x_k}. \tag{3.9}$$

For example, if the norm used in equation (3.5) is the common *euclidean norm*, then a singularity may appear in (3.9). To the author's knowledge, this aspect has never been considered in the literature; therefore the next subsection will clarify this aspect in a more general manner.

[1] the notation $C^k(\Omega)$ indicates a function which is continuous on the domain Ω along with its derivatives up to order k

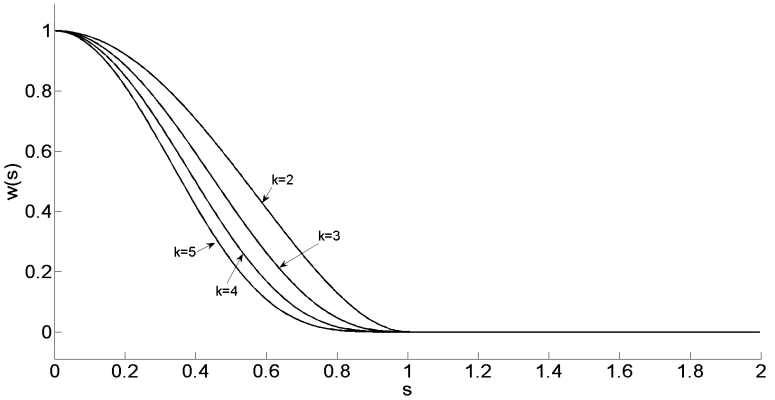


Figure 3.2: Window function (3.8) for different values of k

3.1.2 Window function and its derivatives

Let us consider the weight function w and its first and second derivatives in two dimensions. Defining the following coordinates as in figure 3.3

$$\xi = \frac{x - x_I}{\rho_I} \quad \eta = \frac{y - y_I}{\rho_I} \tag{3.10}$$

and

$$s = \sqrt{\xi^2 + \eta^2} = s(\xi, \eta) \tag{3.11}$$

then

$$\frac{\partial s}{\partial \xi} = \frac{\xi}{s} \qquad \frac{\partial s}{\partial \eta} = \frac{\eta}{s} \tag{3.12}$$

and

$$\frac{\partial^2 s}{\partial \xi^2} = \frac{\eta^2}{s^3}, \qquad \frac{\partial^2 s}{\partial \eta^2} = \frac{\xi^2}{s^3}, \qquad \frac{\partial^2 s}{\partial \xi \partial \eta} = -\frac{\xi\eta}{s^3}. \tag{3.13}$$

If the window functions is a *spline*

$$w(s) = 1 - 6s^2 + 8s^3 - 3s^4 = \sum_{k=0}^{m} a_k s^k \tag{3.14}$$

then

$$\frac{dw}{ds} = -12s + 24s^2 - 12s^3 = \sum_{k=1}^{m} k a_k s^{k-1} \tag{3.15}$$

and

$$\frac{d^2 w}{ds^2} = -12 + 48s - 36s^2 = \sum_{k=2}^{m} k(k-1) a_k s^{k-2}. \tag{3.16}$$

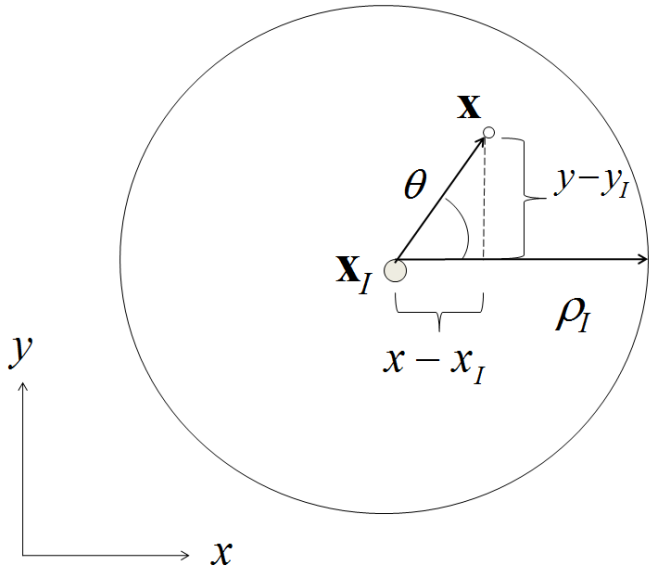


Figure 3.3: Definition of the variables ξ and η

Therefore

$$\frac{\partial w}{\partial \xi} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \xi} = \sum_{k=1}^{m} k a_k s^{k-1}\frac{\eta}{s} \tag{3.17}$$

$$\frac{\partial w}{\partial \eta} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \eta} = \sum_{k=1}^{m} k a_k s^{k-1}\frac{\xi}{s}. \tag{3.18}$$

There is apparently a singularity at $s = 0$, but this *indeterminate form* can be eliminated with the positions

$$\begin{cases} \xi = s\cos(\theta) \\ \eta = s\sin(\theta). \end{cases} \tag{3.19}$$

Hence equations (3.17) and (3.18) become

$$\frac{\partial w}{\partial \xi} = \sum_{k=1}^{m} k a_k s^{k-1}\cos(\theta), \tag{3.20}$$

$$\frac{\partial w}{\partial \eta} = \sum_{k=1}^{m} k a_k s^{k-1}\sin(\theta), \tag{3.21}$$

that evaluated at $s = 0$ and considering that the spline chosen in (3.14) has $a_1 = 0$

$$\frac{\partial w}{\partial \xi}(s = 0) = a_1\cos(\theta) = 0 \qquad \forall \quad \theta, \tag{3.22}$$

$$\frac{\partial w}{\partial \eta}(s=0) = a_1 \sin(\theta) = 0 \qquad \forall \quad \theta, \tag{3.23}$$

$$\frac{\partial^2 w}{\partial \xi^2} = \frac{d^2 w}{ds^2}\left(\frac{\partial s}{\partial \xi}\right)^2 + \frac{dw}{ds}\frac{\partial^2 s}{\partial \xi^2} = \sum_{k=2}^{m} k(k-1)a_k s^{k-2}\frac{\xi^2}{s^2} + \sum_{k=1}^{m} k a_k s^{k-1}\frac{\eta^2}{s^3}, \tag{3.24}$$

$$\frac{\partial^2 w}{\partial \eta^2} = \frac{d^2 w}{ds^2}\left(\frac{\partial s}{\partial \eta}\right)^2 + \frac{dw}{ds}\frac{\partial^2 s}{\partial \eta^2} = \sum_{k=2}^{m} k(k-1)a_k s^{k-2}\frac{\eta^2}{s^2} + \\ + \sum_{k=1}^{m} k a_k s^{k-1}\frac{\xi^2}{s^3}, \tag{3.25}$$

$$\frac{\partial^2 w}{\partial \xi \partial \eta} = \frac{d^2 w}{ds^2}\frac{\partial s}{\partial \eta}\frac{\partial s}{\partial \xi} + \frac{dw}{ds}\frac{\partial^2 s}{\partial \xi \partial \eta} = \sum_{k=2}^{m} k(k-1)a_k s^{k-2}\frac{\xi\eta}{s^2} + \\ - \sum_{k=1}^{m} k a_k s^{k-1}\frac{\xi\eta}{s^3}. \tag{3.26}$$

With the same positions (3.19) equations (3.24), (3.25) and (3.25) become

$$\frac{\partial^2 w}{\partial \xi^2} = \frac{d^2 w}{ds^2}\cos^2(\theta) + \frac{dw}{ds}\frac{\sin^2(\theta)}{s} = \sum_{k=2}^{m} k(k-1)a_k s^{k-2}\cos^2(\theta) + \\ + \sum_{k=1}^{m} k a_k s^{k-2}\sin^2(\theta), \tag{3.27}$$

which evaluated at $s = 0$ is

$$\frac{\partial^2 w}{\partial \xi^2}(s=0) = 2a_2\cos^2(\theta) + \left(\lim_{s\to 0}\frac{a_1}{s} + 2a_2\right)\sin^2(\theta) = 2a_2 \qquad \forall\ \theta \tag{3.28}$$

because $a_1 = 0$.

Analogously

$$\frac{\partial^2 w}{\partial \eta^2} = \frac{d^2 w}{ds^2}\sin^2(\theta) + \frac{dw}{ds}\frac{\cos^2(\theta)}{s} = \sum_{k=2}^{m} k(k-1)a_k s^{k-2}\sin^2(\theta) + \\ + \sum_{k=1}^{m} k a_k s^{k-2}\cos^2(\theta), \tag{3.29}$$

that is

$$\frac{\partial^2 w}{\partial \eta^2}(s=0) = 2a_2 \sin^2(\theta) + \left(\lim_{s\to 0} \frac{a_1}{s} + 2a_2\right) \cos^2(\theta) = 2a_2 \qquad \forall\ \theta \tag{3.30}$$

and

$$\begin{aligned} \frac{\partial^2 w}{\partial \xi \partial \eta} &= \frac{d^2 w}{ds^2} \cos(\theta)\sin(\theta) - \frac{dw}{ds}\frac{\cos(\theta)\sin(\theta)}{s} = \\ &= \sum_{k=2}^{m} k(k-1) a_k s^{k-2} \cos(\theta)\sin(\theta) - \sum_{k=1}^{m} k a_k s^{k-2} \cos(\theta)\sin(\theta) \end{aligned} \tag{3.31}$$

and since $a_1 = 0$

$$\frac{\partial^2 w}{\partial \xi \partial \eta}(s=0) = 2a_2 \cos(\theta)\sin(\theta) - \left(\lim_{s\to 0} \frac{a_1}{s} + 2a_2\right) \cos(\theta)\sin(\theta) = 0 \qquad \forall\ \theta. \tag{3.32}$$

3.1.3 Partition of Unity and Consistency

Shape functions realize a Partition of Unity (PU) if

$$\sum_{I=1}^{N} \phi_I(\mathbf{x}) = 1. \tag{3.33}$$

More generally they realize a $n-th$ order *consistency* if

$$\sum_{I=1}^{N} \phi_I(\mathbf{x}) \mathbf{x}_I{}^{\alpha} = \mathbf{x}^{\alpha} \tag{3.34}$$

where the multi-index notation is used

$$\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_d \end{bmatrix}, \tag{3.35}$$

where $d = \dim(\Omega)$ and $\sum_{i=1}^{d} \alpha_i \leq n$

$$\mathbf{x}^{\alpha} = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_d^{\alpha_d}. \tag{3.36}$$

Equations (3.35) and (3.36) simply define a complete basis according to the dimension of the domain, i.e. 1D, 2D or 3D. Therefore it follows that PU is a $zero-th$ order consistency. Consistency also means that the shape functions are able to reproduce exactly all the polynomials of a complete basis in the dimension d.

3.2 Smooth Particle Hydrodynamics (SPH)

SPH is the forefather of the modern MM Monaghan (1982), Monaghan (1988) and Monaghan (1992). It simply states that

$$u^h(\mathbf{x}) = \int_\Omega w(\mathbf{x}-\mathbf{x}')u(\mathbf{x}')\mathrm{d}\Omega' \tag{3.37}$$

it is basically a *convolution* or a *filtering* of the function u through the kernel. When discretized, equation (3.37) becomes

$$u^h(\mathbf{x}) = \sum_{I=1}^{N} w(\mathbf{x}-\mathbf{x}_I)\Delta V_I U_I, \tag{3.38}$$

where $\Delta V_I = meas(\Omega_I)$ is a measure (i.e. length, area or volume) of Ω_I. Therefore shape functions are given by

$$\phi_I(\mathbf{x}) = w(\mathbf{x}-\mathbf{x}_I)\Delta V_I. \tag{3.39}$$

It can be shown Belytschko *et al.* (1996b) that, choosing a regular arrangement of nodes, SPH shape functions do not realize a partition of unity, precisely at the boundaries, therefore they cannot reproduce exactly even the constant function (figure 3.4).

This is also the reason why these methods are used commonly in *unbounded* domains. If the shape functions are normalized with respect to their sum, then they realize a PU

$${\phi_I}^0(\mathbf{x}) = \frac{\phi_I(\mathbf{x})}{\sum_{I=1}^{N}\phi_I(\mathbf{x})}. \tag{3.40}$$

These functions are called *Shepard's* functions and the superscript 0 means that they have $0-th$ order consistency.

3.3 Reproducing Kernel Method (RKM)

RKM corrects SPH regarding consistency with a *corrected kernel*

$$u^h(\mathbf{x}) = \int_\Omega C(\mathbf{x},\mathbf{x}-\mathbf{x}')w(\mathbf{x}-\mathbf{x}')u(\mathbf{x}')\mathrm{d}\Omega' \tag{3.41}$$

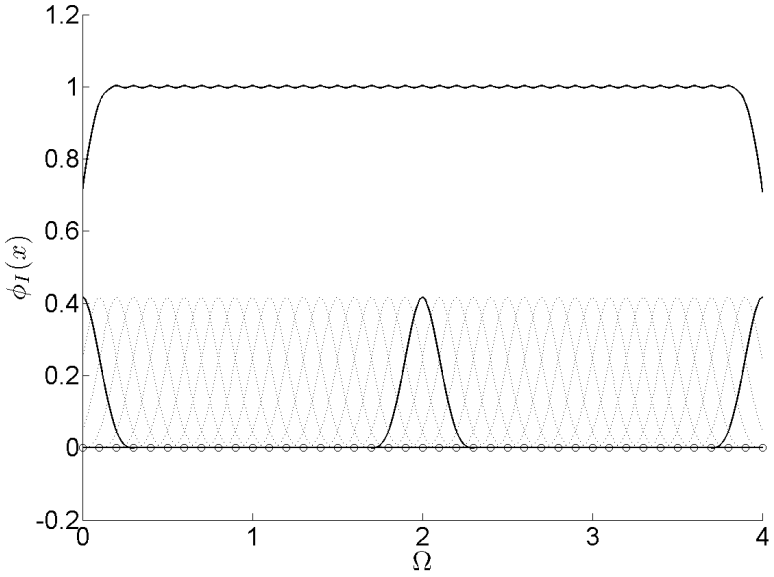


Figure 3.4: SPH shape functions: their sum do not realize a PU

The corrective term $C(\mathbf{x}, \mathbf{x}-\mathbf{x}')$ is obtained by Taylor's series expansion. Let us define a polynomial basis of order n

$$\mathbf{p}(\mathbf{x}) = \left\{ \mathbf{x}^{\alpha} : \sum_{i=1}^{d} \alpha_i \leq n \right\} \tag{3.42}$$

$\mathbf{p}(\mathbf{x})$ is also known as *basis functions*, for example

$$\mathbf{p}^T(\mathbf{x}) = (1, x, x^2, \ldots, x^k) \tag{3.43}$$

in 1D case or

$$\mathbf{p}^T(\mathbf{x}) = (1, x, y, x^2, xy, y^2) \tag{3.44}$$

in 2D case. It can be shown Fries & Matthies (2004) that equation (3.41) is

$$u^h(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{M}(\mathbf{x})^{-1} \int_{\Omega} \mathbf{p}(\mathbf{x}) w(\mathbf{x}-\mathbf{x}') u(\mathbf{x}') \mathrm{d}\Omega' \tag{3.45}$$

where

$$\mathbf{M}(\mathbf{x}) = \int_{\Omega} \mathbf{p}(\mathbf{x}) \mathbf{p}^T(\mathbf{x}) w(\mathbf{x}-\mathbf{x}') \mathrm{d}\Omega' \tag{3.46}$$

is called the *moments matrix*. In fact, if $\mathbf{p}^T(\mathbf{x})$ is substituted in $u(\mathbf{x})$ in equation (3.45) it can be obtained that $u^h(x) = \mathbf{p}^T(\mathbf{x})$ which means that the approximation can reproduce exactly all the polynomials in the basis function.

Sometimes it is preferred a scaled and translated version of equation (3.45), to alleviate the ill-conditioning of the moments matrix.

$$u^h(\mathbf{x}) = \mathbf{p}^T(\mathbf{0})\mathbf{M}(\mathbf{x})^{-1} \int_\Omega \mathbf{p}\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) w\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) u(\mathbf{x}')\mathrm{d}\Omega' \tag{3.47}$$

where

$$\mathbf{M}(\mathbf{x}) = \int_\Omega \mathbf{p}\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) \mathbf{p}^T\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) w\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) \mathrm{d}\Omega'. \tag{3.48}$$

In Barbieri & Meo (2009b), integrals in RKM have been resolved for a general domain of a complex shape.

3.4 Reproducing Kernel Particle Method (RKPM)

RKPM is the discretized version of RKM, in fact considering equations (3.47) and (3.48) and substituting integrals with summation

$$u^h(\mathbf{x}) = \mathbf{p}^T(\mathbf{0})\mathbf{M}(\mathbf{x})^{-1} \sum_{I=1}^{N} \mathbf{p}\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) w\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) \Delta V_I U_I, \tag{3.49}$$

$$\mathbf{M}(\mathbf{x}) = \sum_{I=1}^{N} \mathbf{p}\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) \mathbf{p}^T\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) w\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) \Delta V_I. \tag{3.50}$$

Therefore shape function for node I for RKPM is given by

$$\phi_I(\mathbf{x}) = C_I(\mathbf{x}) w\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right) \Delta V_I, \tag{3.51}$$

$$\underbrace{C_I(\mathbf{x})}_{1\text{x}1} = \underbrace{\mathbf{p}^T(\mathbf{0})}_{1\text{x}k} \underbrace{\mathbf{M}(\mathbf{x})^{-1}}_{k\text{x}k} \underbrace{\mathbf{p}\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right)}_{k\text{x}1}, \tag{3.52}$$

where k is the number of functions in the *basis*. From equation (3.51) is quite evident that $C_I(\mathbf{x})$ is a corrective term for the single weight function centered in $\mathbf{x}_I$. Liu *et al.* (1997a) reported that the corrective term influences the weight

functions only on a strip of length equal to the dilatation parameter around the boundary.

Derivatives for the shape functions (3.51) are given by

$$\frac{\partial \phi_I}{\partial x} = \frac{\partial C_I(\mathbf{x})}{\partial x} w(\mathbf{x}) + C_I(\mathbf{x}) \frac{\partial w(\mathbf{x})}{\partial x}, \tag{3.53}$$

where

$$\frac{\partial C_I}{\partial x} = \mathbf{p}^T(\mathbf{0}) \left[\frac{\partial \mathbf{M}(\mathbf{x})^{-1}}{\partial x} \mathbf{p} + \mathbf{M}(\mathbf{x})^{-1} \frac{\partial \mathbf{p}}{\partial x} \right] \tag{3.54}$$

and

$$\frac{\partial \mathbf{M}(\mathbf{x})^{-1}}{\partial x} = -\mathbf{M}^{-1} \frac{\partial \mathbf{M}}{\partial x} \mathbf{M}^{-1}. \tag{3.55}$$

In Belytschko *et al.* (1996a) it is explained how to speed up the computation of the derivatives with the means of LU factorization.

In equations (3.49), it is clear as the computational burden of MM shape functions is still a factor hindering the diffusion of the meshfree technology. In the next chapter, a solution to this problem is proposed. In fact it is possible to accelerate remarkably the inversion of the moments matrix and its derivatives without using any numerical routines (such as Gauss elimination, LU factorization etc...) and, most importantly, it is possible to avoid such routines at each point of evaluation. This greatly facilitates the implementation and also the portability of the codes on different platforms.

3.5 Moving Least Squares (MLS)

MLS is a least squares procedure where the weights are spatial functions that span all over the domain Ω. As it will be shown later, this procedure leads to the same equations of RKPM. The surprising fact is that, both MLS and RKPM can be derived from the same framework even though they have been originated from different contexts. Indeed, RKPM has its origins in wavelets theory while MLS has its in the approximation theory Lancaster & Salkauskas (1981). Moreover in Liu *et al.* (1997b) and Li & Liu (1996) it is derived a continuous version of MLS, precisely called *Moving Least Squares Reproducing Kernel Method* MLSRKM which is the definitive missing link between the two methods.

The approximation in MLS can be written as

$$u^h(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}(\mathbf{x}). \tag{3.56}$$

Given a set of nodes with coordinates $\mathbf{x}_I$ at which the field variable U_I is known, a weighted $\mathcal{L}_2$ norm can be written

$$J = \sum_{I=1}^{N} w(\mathbf{x} - \mathbf{x}_I) \left[\mathbf{p}^T(\mathbf{x})\mathbf{a}(\mathbf{x}) - U_I\right]^2. \tag{3.57}$$

The minimum of J with respect to the coefficients $\mathbf{a}(\mathbf{x})$ leads to a set of linear equations

$$\mathbf{M}(\mathbf{x})\mathbf{a}(\mathbf{x}) = \mathbf{C}(\mathbf{x})\mathbf{U} \tag{3.58}$$

where

$$\mathbf{M}(\mathbf{x}) = \sum_{I=1}^{N} w(\mathbf{x} - \mathbf{x}_I)\mathbf{p}(\mathbf{x}_I)\mathbf{p}^T(\mathbf{x}_I), \tag{3.59}$$

$$\mathbf{C}(\mathbf{x}) = \begin{bmatrix} w(\mathbf{x} - \mathbf{x}_1)\mathbf{p}(\mathbf{x}_1) & w(\mathbf{x} - \mathbf{x}_2)\mathbf{p}(\mathbf{x}_2) & \dots & w(\mathbf{x} - \mathbf{x}_N)\mathbf{p}(\mathbf{x}_N) \end{bmatrix}, \tag{3.60}$$

$$\mathbf{U} = \begin{bmatrix} U_1 & U_2 & \dots & U_N \end{bmatrix}^T. \tag{3.61}$$

Solving equation (3.58) and substituting in equation (3.56)

$$u^h(\mathbf{x}) = \sum_{I=1}^{N} \mathbf{p}^T(\mathbf{x})\mathbf{M}^{-1}(\mathbf{x})\mathbf{C}_I(\mathbf{x})U_I = \sum_{I=1}^{N} \phi_I(\mathbf{x})U_I, \tag{3.62}$$

where $\mathbf{C}_I(\mathbf{x})$ is the $I - th$ column of $\mathbf{C}$. Thus the shape functions are

$$\phi_I(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{M}^{-1}(\mathbf{x})\mathbf{p}(\mathbf{x}_I)w(\mathbf{x} - \mathbf{x}_I). \tag{3.63}$$

As said previously, this version of shape functions could lead to ill-conditioning of the moments matrix. Therefore a scaled and translated version is commonly preferred, leading to the same equations of RKPM (3.51), (3.52) when $\Delta V_I = 1$. It has been reported in Fries & Matthies (2004) that choosing $\Delta V_I \neq 1$ has no effect on the evaluation of shape functions, thus it can be concluded that MLS and RKPM are practically the same.

To reduce the number of computations in MLS or RKPM, derivatives can be evaluated according to the procedure derived in Belytschko *et al.* (1996a). In fact, if the linear system of equations (3.58) is solved by *LU factor decomposition*, the matrices $\mathbf{L}$ and $\mathbf{U}$ can be stored for subsequent evaluation of the derivatives. Re-writing equation (3.63) in the form

$$\phi_I(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{M}^{-1}(\mathbf{x})\mathbf{p}(\mathbf{x}_I)w(\mathbf{x}-\mathbf{x}_I) = \gamma(\mathbf{x})\mathbf{p}(\mathbf{x}_I)w(\mathbf{x}-\mathbf{x}_I), \tag{3.64}$$

the first derivatives are given by

$$\frac{\partial\phi_I(\mathbf{x})}{\partial x} = \frac{\gamma(\mathbf{x})}{\partial x}\mathbf{p}(\mathbf{x}_I)w(\mathbf{x}-\mathbf{x}_I) + \gamma(\mathbf{x})\mathbf{p}(\mathbf{x}_I)\frac{w(\mathbf{x}-\mathbf{x}_I)}{\partial x}. \tag{3.65}$$

γ is obtained from the following

$$\mathbf{M}(\mathbf{x})\gamma(\mathbf{x}) = \mathbf{p}^T(\mathbf{x}). \tag{3.66}$$

Derivatives of γ can be obtained differentiating equation (3.66)

$$\mathbf{M}(\mathbf{x})\frac{\gamma(\mathbf{x})}{\partial x} = \frac{\partial\mathbf{p}}{\partial x} - \frac{\partial\mathbf{M}(\mathbf{x})}{\partial x}\gamma(\mathbf{x}), \tag{3.67}$$

which can be solved using the same LU decomposition used to solve equation (3.66) and therefore only back-substitutions are necessary to compute equation (3.67).

3.6 Discontinuities in Approximations

In fracture mechanics the presence of a crack introduces a discontinuity in the domain. In FE, elements need to be conforming to the crack line, i.e. no inter-element cracks are allowed, therefore the crack is usually simulated with a very small thickness or by duplicating or splitting the nodes located on the edges. As a consequence, when dealing with propagating cracks, a re-meshing operation is necessary, often requiring human interaction.

MMs overcome all these limitations, because discontinuities can be constructed directly from the shape functions. The first method to introduce discontinuity can be found in Belytschko *et al.* (1994a) and it is called *visibility* criterion. In this method the boundaries of the body and any interior lines of discontinuity are

considered opaque, which means that when the domain of influence for the weight function is constructed the line from a point to a node is imagined to be like a ray of light. If the ray reaches an opaque line, for example an interior discontinuity, the point is not included in the domain of influence (figure 2.1) Thus, nodes on

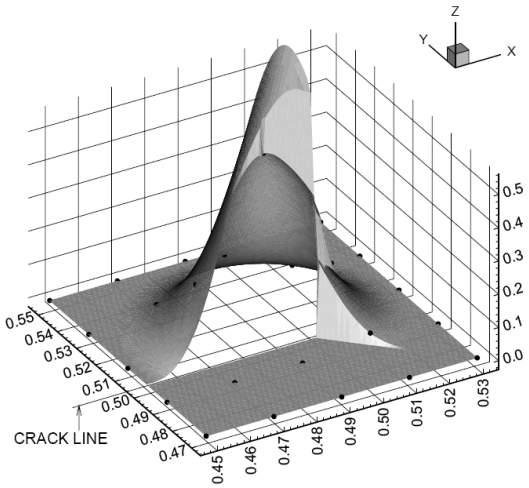


Figure 3.5: Weight function for the visibility criterion: picture taken from Belytschko *et al.* (1994a)

the opposite side of the crack are excluded in the approximation of the displacement field. A major drawback of this method is that non-convex boundaries are not treated properly, since it does not include nodes or point that should instead be included. Moreover, the resulting shape functions are discontinuous, which is usually undesired in a Galerkin procedure, even though convergence of such method has been proved in Krysl & Belytschko (1997).

A modification of this criterion is the *diffraction* method. When the ray encounters a crack line, the distance between the two points is lengthened, including the distance between the point and the crack tip (figure 2.4). Moreover, a smoothing factor λ is included to take into account the density of the nodes or the presence of enrichments. Conversely to the visibility criterion, the diffraction method builds continuous approximations. The weight function distance s is modified as follows

$$s(\mathbf{x}) = \left(\frac{s_1 + s_2(\mathbf{x})}{s_0(\mathbf{x})} \right)^{\lambda} s_0(\mathbf{x}), \tag{3.68}$$

where $s_1 = \|\mathbf{x}_I - \mathbf{x}_c\|$, $s_2(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_c\|$ and $s_0(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_I\|$ where $\mathbf{x}_c$ is the crack tip position and λ is a smoothing factor which for problems with a normal nodal distribution and a linear basis, is chosen as 2. For problems with enrichments, it could be either 1 or 2.

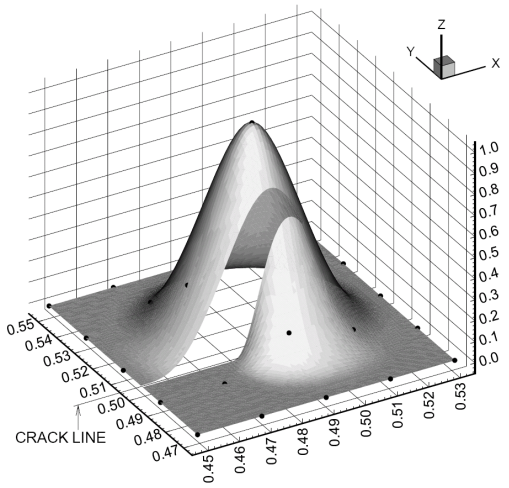


Figure 3.6: Weight function for the transparency criterion: figure taken from Belytschko *et al.* (1996a)

Derivatives of weight function are evaluated accordingly

$$\frac{\partial s}{\partial x_i} = \lambda \left(\frac{s_1 + s_2(\mathbf{x})}{s_0(\mathbf{x})} \right)^{\lambda - 1} \frac{\partial s_2}{\partial x_i} + (1 - \lambda) \left(\frac{s_1 + s_2(\mathbf{x})}{s_0(\mathbf{x})} \right)^{\lambda} \frac{\partial s_0}{\partial x_i}, \tag{3.69}$$

where

$$\frac{\partial s_0}{\partial x_i}(\mathbf{x}) = \frac{x_i - x_{Ii}}{s_0(\mathbf{x})} \tag{3.70}$$

and

$$\frac{\partial s_2}{\partial x_i}(\mathbf{x}) = \frac{x_i - x_{ci}}{s_0(\mathbf{x})}. \tag{3.71}$$

Another technique for obtaining continuous approximations is the transparency method (figure 3.6). The *transparency* is not uniform for the whole length but it changes so that it is completely transparent at the tip and becomes completely opaque a short distance from the tip. In this manner, the range of vision for each node near the crack tip is not suddenly truncated when it reaches the crack tip,

but diminishes smoothly to zero at a short distance from the tip. The length modifies as follows:

$$s(\mathbf{x}) = s_0(\mathbf{x}) + \rho_I \left(\frac{s_c(\mathbf{x})}{\bar{s}_c} \right), \tag{3.72}$$

where $s_c(\mathbf{x})$ is the distance between the crack tip and the intersection point between the ray and the crack line, ρ_I is the dilatation parameter for node I and $\bar{s}_c$ it is a parameter that regulates the transparency transition from the tip to the point of complete opacity. One drawback of the transparency method is that it does not work well when nodes are placed too close to the crack surface.

Other methods employed are the *see-through* method and most recently, the *spiral weight method* Muravin & Turkel (2006b) takes into consideration the advantages and drawbacks of the diffraction method. It is claimed in this method that the use of this weight function can accurately solve for the stresses even when using a linear basis with no enrichments.

3.7 Intrinsic Enrichments

Intrinsic enrichment consists simply in adding functions to the basis that resemble the known solution of a particular problem. For example, for linear elastic fracture mechanics in two dimensions, one can include the asymptotic near-tip displacement field

$$\mathbf{p}^T(\mathbf{x}) = \begin{bmatrix} 1 & x & y & \sqrt{r}\cos\frac{\theta}{2} & \sqrt{r}\sin\frac{\theta}{2} & \sqrt{r}\cos\frac{\theta}{2}\sin\theta & \sqrt{r}\sin\frac{\theta}{2}\sin\theta \end{bmatrix}, \tag{3.73}$$

where

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2} \tag{3.74}$$

and

$$\theta = \text{atan2}(y - y_c, x - x_c), \tag{3.75}$$

where $\theta \in [-\pi/2, \pi/2]$ is the angle from the segment between the point $\mathbf{x}$ and the crack tip $\mathbf{x}_c$ and the tangent of the crack line (figure 3.8).

It can be proved that these augmented functions can *reproduce* exactly the near-tip asymptotic displacement field

$$u(\mathbf{x}) = \frac{k_1}{2\mu}\sqrt{\frac{r}{2\pi}}\cos\left(\frac{\theta}{2}\right)\left[\kappa - 1 + 2\sin^2\left(\frac{\theta}{2}\right)\right], \tag{3.76a}$$

$$v(\mathbf{x}) = \frac{k_1}{2\mu}\sqrt{\frac{r}{2\pi}}\sin\left(\frac{\theta}{2}\right)\left[\kappa - 1 + 2\cos^2\left(\frac{\theta}{2}\right)\right], \tag{3.76b}$$

where κ is the *Kolosov* constant

$$\kappa = \begin{cases} 3 - 4\nu & \text{plane strain} \\ \frac{(3-\nu)}{1+\nu} & \text{plane stress} \end{cases}, \tag{3.77}$$

μ is the shear modulus and k_1 is the stress intensity factor for mode I.

Even though there are no extra unknowns, for multiple cracks, four additional terms need to be added to the basis for each crack. This increase the size of the moments matrix, which means more computational effort and more dense distribution of nodes, because each support must contain at least a number of nodes equal to the size of the basis Huerta & Fernandez-Mendez (2000).

There are also ill-conditioning issues when the basis contains non-polynomial basis Belytschko & Fleming (1999). A method for coupling approximation with enriched and polynomial basis is proposed in Belytschko & Fleming (1999) where

$$u^h(\mathbf{x}) = Ru^{enr}(\mathbf{x}) + (1 - R)u^{lin}(\mathbf{x}), \tag{3.78}$$

where $u^{enr}(\mathbf{x})$ is the enriched approximation, $u^{lin}(\mathbf{x})$ is the linear basis approximation and R is a ramp function which is equal to unity on the enriched boundary of the coupling region and equal to zero on the linear boundary of the coupling region. The same approach is used in Belytschko *et al.* (1996b) to couple FE and MM.

3.8 Extrinsic Enrichment

Extrinsic enrichments are based on the broader concept of *Partition of Unity* Duarte & Oden (1996) Melenk & Babuška (1996). There are two different types of extrinsic enrichment, namely *extrinsic global enrichment* and *extrinsic local*

enrichment. The global enrichment consists in adding functions to the approximation that are evaluated on all the Gaussian points of the domain, whereas an extrinsic local enrichment limits these extra functions only in specific parts of the domain. It will be shown in the next sections that the second approach is achieved exploiting the PU property of the shape functions.

3.8.1 Extrinsic Global Enrichment

In the global enrichment the approximation $u^h(\mathbf{x})$ for each component of the displacement is written as

$$u_i^h(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}_i(\mathbf{x}) + \sum_{j=1}^{n_c} \alpha^j Q_i^j(\mathbf{x}) \quad i = 1, \ldots, n_c, \tag{3.79}$$

where n_c is the number of the cracks, $\mathbf{a}_i(\mathbf{x})$ coefficients of the polynomial basis, α^j is an unknown associated with the $j-th$ crack and $Q_i^j(\mathbf{x})$ are near-tip displacement field for mode I

$$Q_1(\mathbf{x}) = \frac{1}{2\mu}\sqrt{\frac{r}{2\pi}}\cos\left(\frac{\theta}{2}\right)\left[\kappa - 1 + 2\sin^2\left(\frac{\theta}{2}\right)\right], \tag{3.80a}$$

$$Q_2(\mathbf{x}) = \frac{1}{2\mu}\sqrt{\frac{r}{2\pi}}\sin\left(\frac{\theta}{2}\right)\left[\kappa - 1 + 2\cos^2\left(\frac{\theta}{2}\right)\right]. \tag{3.80b}$$

Using MLS procedure, in terms of shape functions, equation (3.79) becomes

$$u_i^h(\mathbf{x}) = \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_{Ii} + \sum_{j=1}^{n_c} \alpha^j \left[Q_i^j(\mathbf{x}) - \sum_{I=1}^{N} \phi_I(\mathbf{x}) Q_i^j(\mathbf{x}_I) \right], \tag{3.81}$$

where $\phi_I(\mathbf{x})$ is the shape function defined in equation (3.63). It should be noted that in this method there are only n_c additional unknowns of the problem. This means that if only 1 crack exists, there is just one more unknown per component of displacements, which is quite advantageous. Moreover, for crack problems, in Nguyen *et al.* (2008) is reported that extrinsic global enrichments for crack problems are far superior in terms of accuracy on the stress intensity factors with respect to the local enrichment and can be directly obtained without considering the J-integral. However, the computational cost is higher since the enrichment

is applied in the entire domain. The extrinsic local enrichment based on the PU is only slightly less accurate but computationally more efficient, since the enrichment is applied locally and little modifications to existing codes are needed to turn a standard meshfree code into an *enriched* one.

3.8.2 Extrinsic Local Enrichment

The *extrinsic local enrichment* or *extrinsic PU enrichment* instead makes use of the Shepard's function (3.40). This approach will be later used to simulate delaminations in composite materials.

For the sake of the clarity, let us start with a simple example.

$$u^h(x) = \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I + \Psi(\mathbf{x}) \tag{3.82}$$

where $\Psi(\mathbf{x})$ is a generic enrichment function, for example the near-tip crack enrichment or a strong discontinuity like the Heaviside function.

It may seem that (3.82) is similar to the global extrinsic equation. Indeed, to localize the enrichment, one simply has to consider (3.33) and multiply 1 to $\Psi(\mathbf{x})$

$$\begin{aligned} u^h(x) &= \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I + \Psi(\mathbf{x}) \underbrace{1}_{\sum_{I=1}^{N_e} \phi_I^0(\mathbf{x}) a_I} = \\ &= \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I + \Psi(\mathbf{x}) \sum_{J=1}^{N_e} \phi_J^0(\mathbf{x}) a_J = \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I + \sum_{J=1}^{N_e} \phi_J^0(\mathbf{x}) \Psi(\mathbf{x}) a_J, \end{aligned} \tag{3.83}$$

where the superscript 0 means *zero-th order consistency*. The number N_e is the number of enriched nodes and a_J are added unknowns to these nodes. If all the a_J are equal to 1, then, for the PU, the (3.83) becomes (3.82) and therefore $\Psi(\mathbf{x})$ is reproduced exactly. The locality of the enrichment stays in the term $\phi_J^0(\mathbf{x})\Psi(\mathbf{x})$. In fact, the enriched nodes are chosen as the nodes whose support contains the enrichment. If it is a discontinuity, then the enriched nodes are the ones whose support is cut by the discontinuity. If $\Psi(\mathbf{x})$ is a near-tip crack enrichment, then the enriched nodes are the ones whose support contains the crack tip. Since the ϕ_J^0 are compact support functions, the enrichment $\Psi(\mathbf{x})$ does not have to be

evaluated to all the $\mathbf{x} \in \Omega$ but only to all the (Gaussian) points $\mathbf{x}$ belonging to the N_e supports of the ϕ_J^0.

Duarte & Oden (1996) pointed out that also standard MLS shape functions (3.63) are PU; therefore in equation (3.82) Shepard's functions could be replaced by $\phi_I^n(\mathbf{x})$ where the superscript n denotes any shape functions satisfying a PU of order n.

Apart from the minor computational cost of this formulation, another advantage is the facilitation of *hp-adaptivity*, since the enhanced functions may vary from node to node. In fact, for crack problems, it is possible to use the following enrichment Ventura *et al.* (2002)

$$\mathbf{u}^h(\mathbf{x}) = \sum_I^N \phi_I(\mathbf{x})\mathbf{U}_I + \sum_J^{N_c} \phi_j(\mathbf{x})\mathcal{H}(\mathbf{x})\mathbf{a}_j + \sum_k^{N_b} \phi_k(\mathbf{x}) \sum_{\alpha=1}^{4} \mathcal{B}_\alpha(\mathbf{x})\mathbf{b}_{\alpha k}, \tag{3.84}$$

where $\mathbf{a}_j$ and $\mathbf{b}_{\alpha k}$ are added unknowns, $\mathcal{H}(\mathbf{x})$ is the Heaviside function, N_c is the number of nodes whose support is completely cut by the crack, N_b is the number of nodes whose support contains the crack tip and $\mathcal{B}_\alpha$ are the branch functions (figure 3.7)

$$\mathcal{B}^T = \left[\sqrt(r)\cos\left(\tfrac{\theta}{2}\right) \quad \sqrt(r)\sin\left(\tfrac{\theta}{2}\right) \quad \sqrt(r)\sin\left(\tfrac{\theta}{2}\right)\sin(\theta) \quad \sqrt(r)\cos\left(\tfrac{\theta}{2}\right)\sin(\theta)\right] \tag{3.85}$$

where

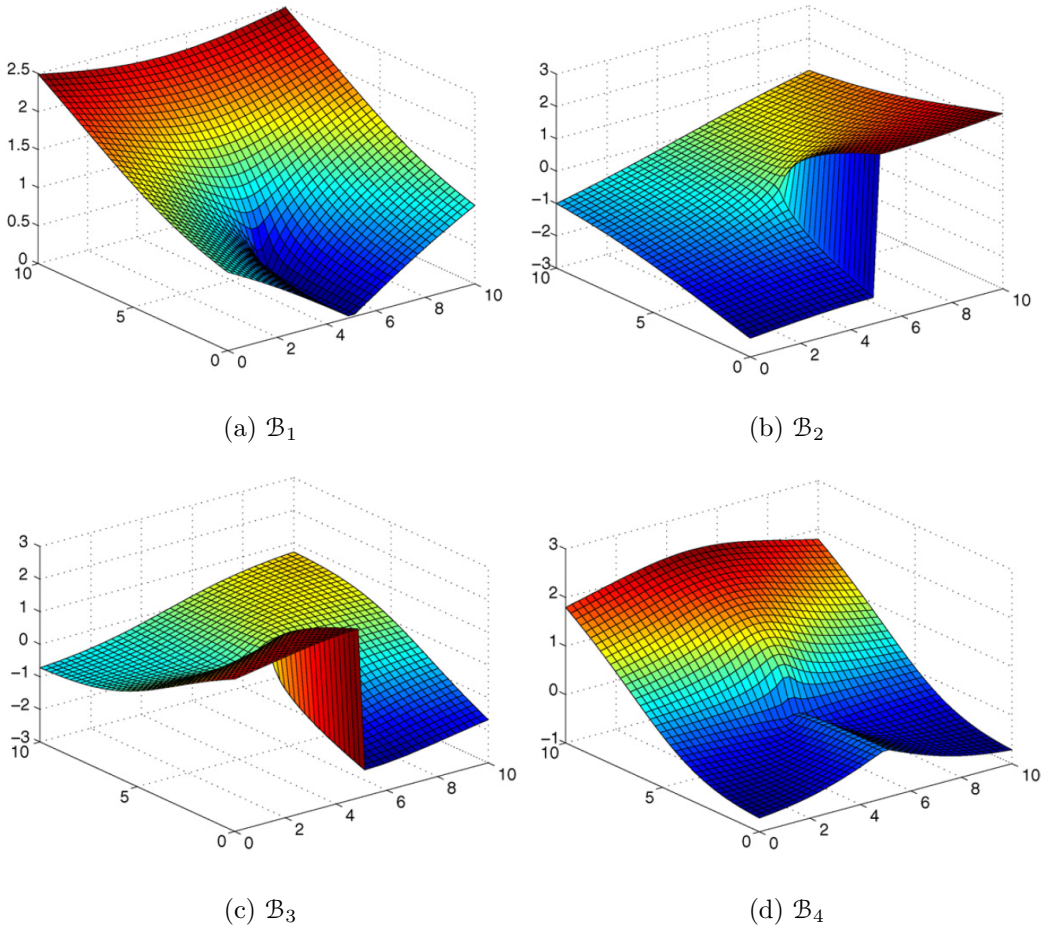
$$r = \sqrt{x^2 + y^2} \tag{3.86}$$

$$\theta = \frac{y}{\sqrt{x^2 + y^2} + x} \tag{3.87}$$

as shown in figure 3.8.

For completeness, here the derivatives of the branch functions are also reported

$$\frac{\partial(r,\theta)}{\partial(x,y)} = \begin{bmatrix} x/r & y/r \\ -y/(2r^2) & x/(2r^2) \end{bmatrix} \tag{3.88}$$



(a) $\mathcal{B}_1$ (b) $\mathcal{B}_2$

(c) $\mathcal{B}_3$ (d) $\mathcal{B}_4$

Figure 3.7: Branch functions

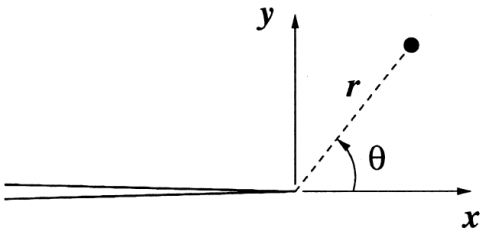


Figure 3.8: Near crack-tip polar coordinates

$$\frac{\partial \mathcal{B}}{\partial(r,\theta)} = \begin{bmatrix} \frac{1}{2\sqrt{r}}\cos\left(\frac{\theta}{2}\right) & -\sqrt{r}\sin\left(\frac{\theta}{2}\right) \\ \frac{1}{2\sqrt{r}}\sin\left(\frac{\theta}{2}\right) & -\sqrt{r}\cos\left(\frac{\theta}{2}\right) \\ \frac{1}{2\sqrt{r}}\sin^2\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right) & \sqrt{r}\left(3\cos^2\left(\frac{\theta}{2}\right)-1\right)\left(\sin\left(\frac{\theta}{2}\right)\right) \\ \frac{1}{2\sqrt{r}}\cos^2\left(\frac{\theta}{2}\right)\sin\left(\frac{\theta}{2}\right) & \sqrt{r}\left(3\cos^2\left(\frac{\theta}{2}\right)-2\right)\left(\cos\left(\frac{\theta}{2}\right)\right) \end{bmatrix}. \tag{3.89}$$

Therefore

$$\frac{\partial \mathcal{B}}{\partial(x,y)} = \frac{\partial \mathcal{B}}{\partial(r,\theta)}\frac{\partial(r,\theta)}{\partial(x,y)}, \tag{3.90}$$

$$\frac{\partial \mathcal{B}_1}{\partial x} = \frac{1}{2}\cos\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}} + \frac{1}{4}\sin\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}}, \tag{3.91}$$

$$\frac{\partial \mathcal{B}_1}{\partial y} = \frac{1}{2}\cos\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}} - \frac{1}{4}\sin\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}}, \tag{3.92}$$

$$\frac{\partial \mathcal{B}_2}{\partial x} = \frac{1}{2}\sin\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}} - \frac{1}{4}\cos\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}}, \tag{3.93}$$

$$\frac{\partial \mathcal{B}_2}{\partial y} = \frac{1}{2}\sin\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}} + \frac{1}{4}\cos\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}}, \tag{3.94}$$

$$\frac{\partial \mathcal{B}_3}{\partial x} = \sin^2\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}} - \frac{1}{2}\left(3\cos^2\left(\frac{\theta}{2}\right)-1\right)\sin\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}}, \tag{3.95}$$

$$\frac{\partial \mathcal{B}_3}{\partial y} = \sin^2\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}} + \frac{1}{2}\left(3\cos^2\left(\frac{\theta}{2}\right)-1\right)\sin\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}}, \tag{3.96}$$

$$\frac{\partial \mathcal{B}_4}{\partial x} = \sin^2\left(\frac{\theta}{2}\right)\cos\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}} - \frac{1}{2}\left(3\cos^2\left(\frac{\theta}{2}\right)-1\right)\sin\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}}, \tag{3.97}$$

$$\frac{\partial \mathcal{B}_4}{\partial y} = \cos^2\left(\frac{\theta}{2}\right)\sin\left(\frac{\theta}{2}\right)\frac{y}{r^{3/2}} + \frac{1}{2}\left(3\cos^2\left(\frac{\theta}{2}\right)-2\right)\cos\left(\frac{\theta}{2}\right)\frac{x}{r^{3/2}}. \tag{3.98}$$

3.9 Element-Free Galerkin (EFG)

So far only approximations to the unknown function have been considered. In order to solve a differential equation, methods like *variational formulations* or collocative methods (like finite difference) must be used. The EFG method Belytschko *et al.* (1994b) is based on the weak variational form of the differential equations of the elasticity along with the boundary conditions. For example, in dynamic problems,

$$\int_{\Omega} \delta\epsilon^T \sigma d\Omega - \int_{\Omega} \delta\mathbf{u}^T \mathbf{b} d\Omega - \int_{\Gamma} \delta\mathbf{u}^T \mathbf{t} d\Gamma + \int_{\Omega} \delta\mathbf{u}^T \rho\ddot{\mathbf{u}} d\Omega + \alpha \int_{\Gamma_u} \delta\mathbf{G}(\mathbf{u})^T \mathbf{G}(\mathbf{u}) d\Gamma_u = 0, \tag{3.99}$$

where Ω is the entire domain of the solid, Γ its whole boundary, Γ_u the part of boundary where are imposed the *essential boundary conditions*, σ is the vectorized stress tensor, ϵ is the vectorized strain tensor, ρ the mass density, $\mathbf{u}$ is the displacements vector, $\mathbf{b}$ and $\mathbf{t}$ are respectively the body and the surface forces and $\mathbf{G}(\mathbf{u})$ the vector of the essential boundary conditions, for example in $2D$

$$\mathbf{G}(\mathbf{u}) = \begin{bmatrix} u - \bar{u} \\ v - \bar{v} \end{bmatrix} \qquad \forall \mathbf{x} \in \Gamma_u. \tag{3.100}$$

The parameter α is a *penalty parameter* Zienckiewicz & Taylor (1994) used to impose essential boundary conditions.

In fact, because of the non-interpolating nature of MLS, essential boundary conditions cannot be directly imposed on the nodes. Thus, a penalty method is needed to enforce boundary conditions on displacements. Other methods used to impose essential boundary conditions are *Lagrange Multipliers* or *coupling with FE*. To the author's opinion, *penalty method* must be preferred to *Lagrange Multipliers* since no extra unknowns are introduced (the Lagrange multipliers λ) and also no zeros appear in the diagonal of the resulting stiffness matrix. The introduction of extra zeros could cause spurious zero-modes in a dynamic analysis. Also, the symmetry of the stiffness matrix is destroyed by the introduction of the multipliers, with disadvantages in the numerical solutions of the algebraic system of equations. Moreover *penalty method* may be easily used to impose contacts between two or more bodies and also to separate sub-domains of a whole body. An example of the power of the *penalty method* can be found in Barbieri &

Meo (2008) and Barbieri & Meo (2009a) where the delamination in a composite material is studied in conjunction to a *cohesive zone model*. A generalization of the *cohesive zone model* applied to meshless method can be found in Sun *et al.* (2007) and it will be presented in the next chapters.

Substitution of equation (3.63) in equation (3.99) leads to the discretized equation of motion

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{f}(t), \tag{3.101}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\mathbf{K}$ is the stiffness matrix and $\mathbf{f}$ is the vector of generalized forces. It should be remarked that $\mathbf{U}$ is a "fictitious" nodal displacements vector because of equation (3.2). Mass and stiffness matrices are given by

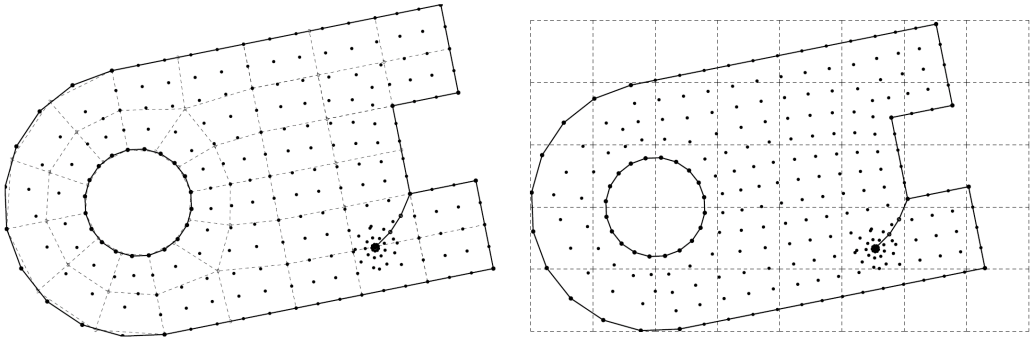
$$\mathbf{M}_{ij} = \int_{\Omega} \rho\phi_i\phi_j d\Omega, \tag{3.102}$$

$$\mathbf{K}_{ij} = \int_{\Omega} \mathbf{B}_i^T\mathbf{D}\mathbf{B}_j d\Omega + \alpha \int_{\Gamma_u} \phi_i\phi_j d\Gamma_u, \tag{3.103}$$

where $\mathbf{B}$ is the two-dimensional strain differential operator matrix applied to all shape functions ϕ_I, $\mathbf{D}$ is the elastic constitutive matrix and $\mathbf{f}(t)$ generalized forces are given by

$$\mathbf{f}_i(t) = \int_{\Omega} \phi_i\mathbf{b}d\Omega + \int_{\Gamma} \phi_i\mathbf{t}d\Gamma + \alpha \int_{\Gamma_u} \phi_i\bar{\mathbf{u}}d\Gamma_u. \tag{3.104}$$

Because of the complex nature of the shape functions, integrals in equations (3.102), (3.103) and (3.104) have to be evaluated numerically. Since mesh-free methods have no intrinsic subdivision like finite elements, it is necessary to introduce a subdivision of the domain. Normally two approaches are used, *background elements* (figure 3.9a) or *cell quadrature* (figure 3.9b). In the former one, quadrature is performed over a subdivision of the domain built by a FE mesh generator and the vertices are taken as initial array nodes for MLS approximation, while in the latter an octree cell structure, independent of the domain, is used. Each integration point is checked as to whether or not it lies inside the domain; points which lie outside are neglected. One may think that cell quadrature is less performing than background elements, because large errors are expected when surfaces pass through the cell and at the boundaries; nonetheless, in Belytschko & Fleming



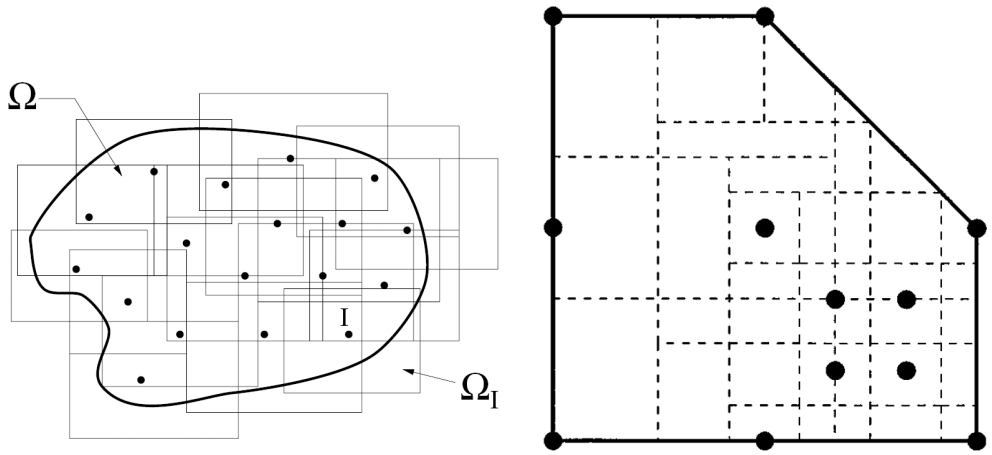
(a) Background elements (b) Cell quadrature

Figure 3.9: Quadrature methods in MM; pictures taken from Belytschko *et al.* (1996b)

(1999) it is reported that no significant difference can be found using the two different types.

In Dolbow & Belytschko (1999) is proposed a *bounding box technique* which takes into account the overlapping supports. This technique seems reasonable for tensor-product kernels (figure 3.10a), where the product of 2 shape functions in different nodes is different from zero only where the supports overlap each other. The minimal regions of these overlapping determine the bounding box technique in figure 3.10b. Therefore such integration method is expected to greatly reduce the numerical errors due to numerical integration, whenever a tensor product kernel is used. Moreover, it is usually recommended to use *Gaussian* integration over these cells rather than *trapezoidal* integration, since *Gaussian* is more accurate than the trapezoidal for the same number of integration points. In Dolbow & Belytschko (1999) it is also mentioned that more smaller integration cells with less integration points is a better choice than larger integration cells with higher order accuracy, because shape functions usually vary quite rapidly over a small portion of the domain. The use of cells or elements for the numerical integration might sound incoherent, since MMs are expected not to use any sort of mesh. This is indeed a quite controversial topic, that led some researchers to distinguish between *truly* meshless methods and not truly MM Atluri & Zhu (1998).

In fact, to create truly MM, one needs to construct *local* weak formulations



(a) Tensor product kernels (b) Bounding box technique

Figure 3.10: Bounding box integration; pictures taken from Dolbow & Belytschko (1999)

Atluri & Zhu (1998), while EFG is a *global* weak formulation.

In response to this paradox, Liu (2003) said:

> *"One may ask now, why do we not simply use the conventional finite element method (FEM) with triangular elements, which has been fully developed? The answer is simply the well-known fact that the FEM results using triangular elements are very poor. This is due to the poor quality of the linear shape functions constructed based on three-node element confined interpolation schemes. In meshfree methods, however, the interpolation/ approximation is not cell/element confined. Basically, it can "freely," "automatically," and "dynamically" choose surrounding nodes to construct the shape functions. This freedom of node selection enabled by meshfree technology is truly powerful. Based on this argument, the difference between FEM and meshfree methods is essentially the difference in the interpolation scheme, and this is also the reason the interpolation technique is the key to the success of the meshfree methods."*

Chapter 4

Improvements to the Method

The major factors hindering the spreading of the meshfree methods are of practical nature. Indeed, for equation (3.49), it is necessary to invert a matrix $\mathbf{M}$, called the *moment matrix*, for each point of interest. If the number of points is large, then this operation must be repeated several times, raising the computational cost.

Another issue regards the search for points located within the support radius. In fact, for equation (3.5), meshfree shape functions are *compact support* functions, which means that they are zero outside a ball of radius ρ_I centered in the $I-th$ node. This means that, in equation (3.49), the summation can be limited to the nodes that have $\mathbf{x}$ in their support. Hence, to evaluate the shape functions, the following operations must be performed at every point $\mathbf{x}$ (figure 4.1) Krysl & Belytschko (2001) and Fries & Matthies (2004):

- search for neighbors within the set of nodes
- calculate the moment matrix (see equation (3.50))
- invert the moment matrix
- apply the corrective term to the weight function (see equation (3.51))

Also derivatives are usually of interest in Galerkin formulations.

Since these operations are carried out within a loop over the Gaussian points, and since the Gaussian points are always more than the nodes, it is easy to see that the overall code is incredibly slowed down.

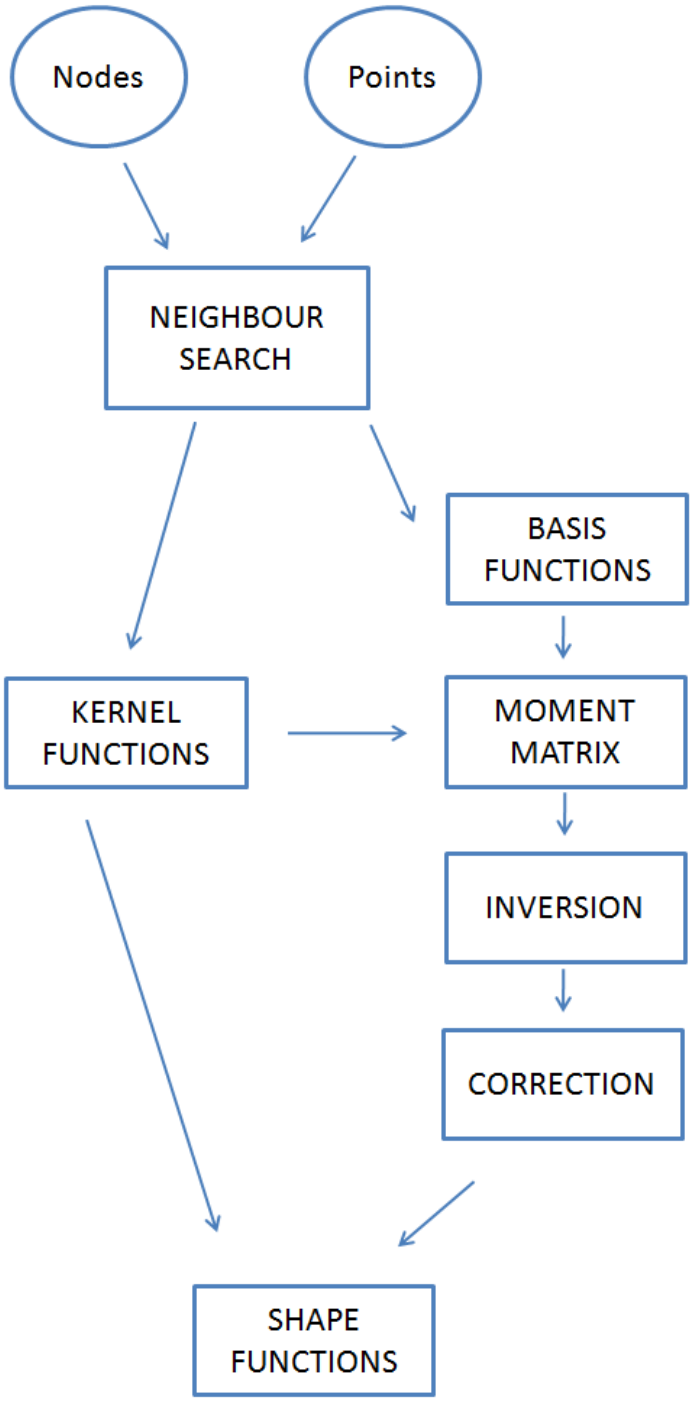


Figure 4.1: Flowchart of the *classic* algorithm in meshfree methods

In this chapter some remedies to implementation issues in MM will be presented, while in the next chapter the complete code will be described.

As a general guideline, the double-looping over the nodes and the Gaussian points should be avoided as much as possible. This is true for every programming language but it is more evident for `Matlab` , where significant speeds-up can be achieved with loop *vectorization*. Moreover, the Gaussian points are usually are more numerous than the nodes, therefore instead of looping over the Gaussian points and looking for the nodes, it is more convenient to focus on nodes and search for the Gaussian points included in the support.

In the following sections the typewriter character `A` will be used to indicate the computer variable, while $\mathbf{A}$ will indicate the mathematical entity (vector, matrix, etc...).

4.1 *kd*-Trees for Neighbour Search

Meshfree shape functions are *compact support functions*, as stated in equation (3.5) and properties (3.6a) and (3.6b).

The purpose of the compactness is to *discretize* the domain, as in FE, i.e. to give a local character to the approximation. As a result, stiffness matrix is *sparse*, i.e. with most entries equal to zero. The compactness of the shape functions can be exploited to better store the matrix values and to efficiently evaluate the integrals in Galerkin formulations. In fact, figure 4.2 shows the sparsity pattern arising from a typical meshfree discretization over a line segment. If n_g is the number of evaluation points (rows) and n_s the number of nodes, the meshfree shape functions resulting from a discretization can be stored in a $n_g \times n_s$ matrix `PHI`.

For example, for a fixed column [1] `PHI` contains the values of the $j-th$ shape function on all the points that belong to the support of the $j-th$ node.

$$\texttt{PHI}(:, \texttt{j}) \qquad j = 1, \ldots, n_s. \tag{4.1}$$

[1] it is used the `Matlab` notation where the colon : in a matrix means variation over that dimension

Since on the same evaluation point there should be enough nodes for the meshfree approximation [1], analogously, for a fixed row `PHI` contains the values of the shape functions of the nodes that have the $i-th$ evaluation point within their support.

$$\texttt{PHI(i,:)} \qquad i = 1, \ldots, n_g. \tag{4.2}$$

The sparsity diagram 4.2 is a representation of the non-zero values of `PHI`, where a blue dot corresponds to a non-null entry. Firstly, it can be seen how the number of evaluation points is usually much larger than the number of nodes. These evaluation points usually are Gaussian points, if the variational weak form is used. Secondly, it can be clearly seen that the vast majority of entry is zero and concentrated on the main diagonal.

This reflects the fact that for each node, only the nearest Gaussian points are considered. When two supports for two different nodes overlap, then there is a non-zero entry for the stiffness matrix, otherwise the correspondent entry will be null. Thus, it seems reasonable not to store the shape functions matrix as a full matrix, but rather as sparse matrix. The only values stored are then the row-index, the column-index and the non-zero entry. A full matrix would have had all the values, even the zero ones, arranged in a *table-like* manner.

As an example of the amount of memory saved, a factor called density d is normally calculated for sparse matrix. It is the ratio of the number of non-zero values n_{nz} and the total number of elements of the matrix, i.e. the product of the total number of rows r for the total number of columns c. For meshfree shape functions, this factor could be around 6%, depending of course on the dilatation parameter.

$$d = \frac{n_{nz}}{rc}. \tag{4.3}$$

In order to obtain this sparse matrix, it is crucial an algorithm capable of searching the evaluation points within a certain radius (figure 4.3).

Given a set of n_s approximation nodes (variable `GRIDSET`)

$$S_1 = \{\mathbf{x}_J : \ J = 1, \ldots, n_s\} \subset \mathbb{R}^k \qquad k = 1, 2, 3 \tag{4.4}$$

[1] to avoid ill-conditioning of the moment matrix Fries & Matthies (2004) and Nguyen *et al.* (2008)

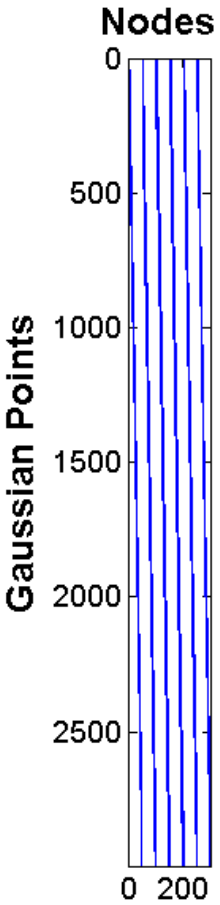


Figure 4.2: Sparsity pattern for meshfree shape functions on a segment

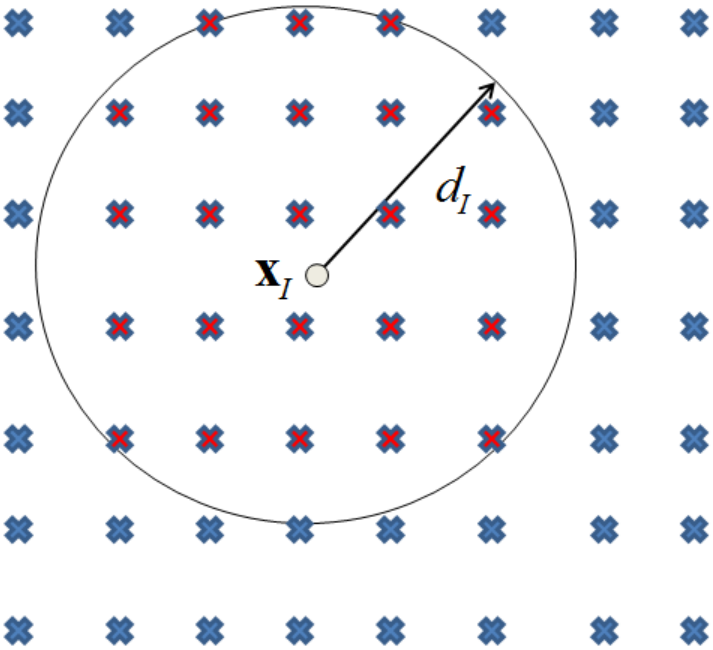


Figure 4.3: Neighbour Search: red crosses: evaluation points inside the support; blue crosses: evaluation points outside the support

and a second set of n_g evaluation points (variable `GGRID`[1])

$$S_2 = \{\mathbf{x}_I : \ I = 1, \ldots, n_g\} \subset \mathbb{R}^k \qquad k = 1, 2, 3 \tag{4.5}$$

the complete *distance matrix* $\mathbf{S}$ would be a matrix $n_g \times n_s$ where

$$\mathbf{S}_{IJ} = \frac{|\mathbf{x}_I - \mathbf{x}_J|}{\rho_J}, \tag{4.6}$$

where ρ_J is the dilatation parameter for the *J-th* node.

After $\mathbf{S}$ matrix is constructed, for equation (3.5) only the values less than or equal to one must be considered. Let us call n_{nz} the number of elements in $\mathbf{S}$ satisfying

$$\mathbf{s} = \{I \in 1, \ldots n_g, \ J \in 1, \ldots n_s : \quad \mathbf{S}_{IJ} \leq 1\} \tag{4.7}$$

then, the output of such procedure would eventually be 3 vectors of length n_{nz}, $\mathbf{i_g}$, $\mathbf{j_s}$ and $\mathbf{s}$ such that (figure 4.4)

$$\mathbf{S}_{\mathbf{i_g}_i, \mathbf{j_s}_i} = \mathbf{s}_i \quad i = 1, \ldots, n_{nz}. \tag{4.8}$$

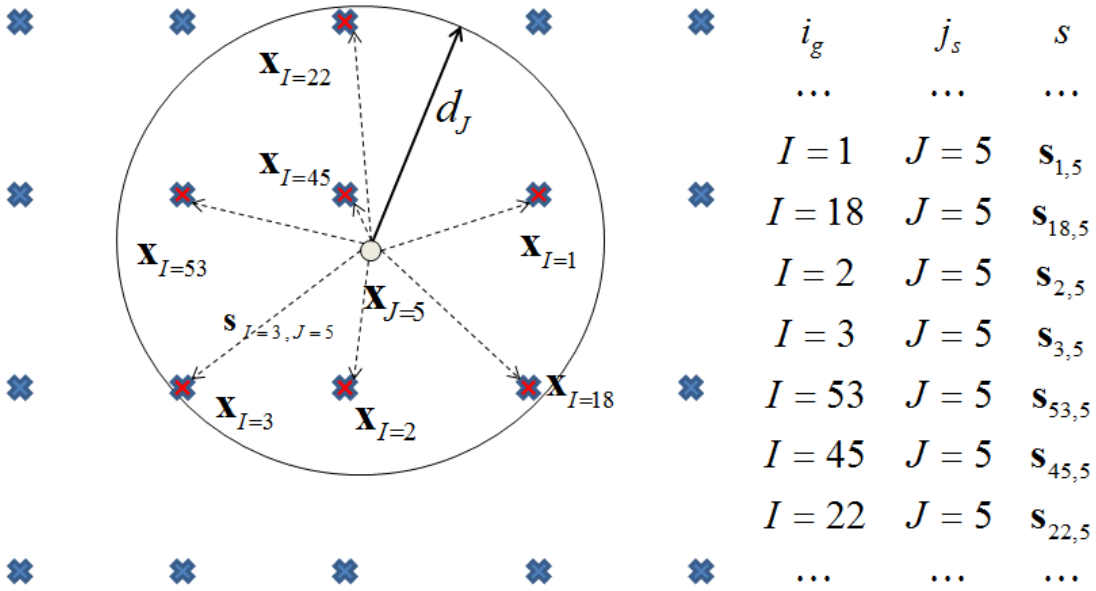


Figure 4.4: Neighbour Search: example of the mapping in equation (4.8)

This is a well-known problem in the field of the computational geometry and it is called *neighbour search* or *range query*.

A *naive* implementation would loop over the Gaussian points S_2, loop again over the nodes S_1, compare the distance with the dilatation parameter and then decide whether a point belongs to the node or not. This *brute force* algorithm is said to have a computational cost $\mathcal{O}(N^2)$, which means that the number of operations goes with the square of the number N of the elements. If N is a large number, then it is easy to understand that the computational cost becomes prohibitive. Yet, this algorithm for its simplicity is widely used in the meshfree community as affirmed in Idelsohn & Oñate (2006) and reported in several papers Belytschko *et al.* (1996b) , Dolbow & Belytschko (1998), Krysl & Belytschko (2001) and Nguyen *et al.* (2008).

The brute force algorithm is nonetheless inefficient when the number of nodes notably grows. It is then especially not recommended for large scale simulations. The neighbour search is indeed recognized as one of the greatest bottle-neck for the meshfree technology. For a more efficient code, improvements should be

[1] in principle different from `GRIDSET`, but not necessarily

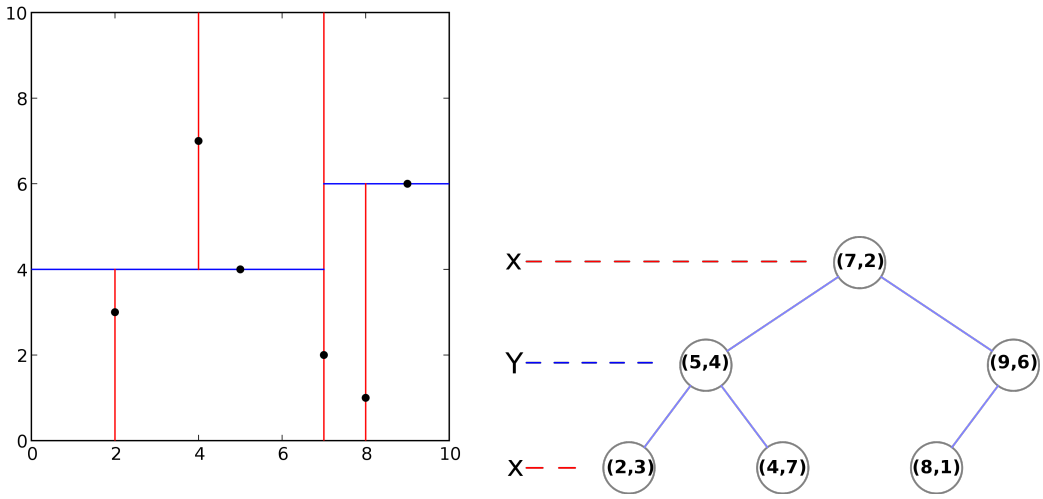
directed also in this sense.

An attempt towards this scope is reported in Liu (2003) where an algorithm called *bucket* is used to reduce the range of nodes over the search is performed. The algorithm allows a maximum number of nodes to fall within each bucket. If the number of nodes in the bucket exceeds a certain pre-fixed value, then the bucket is split in two parts. The subdivision is iterated until every bucket has a number of nodes inferior to a defined one. This is roughly the concept behind a more powerful tool, i.e. *kd-tree*.

A *kd-tree* is a generalization of binary trees to $k-$dimensional data. Binary trees are studied in *computer science* as *data structure* that emulates a hierarchical tree structure with a set of linked nodes. Every node is connected to at most 2 other nodes, called *children*. If each node is connected to 4 children, then the tree is called a *quadtree*, if connected to 8 children the tree is a *octree*. The quadtree idea is identical to the *bucket* idea, and in fact it is normally used to to partition a two dimensional space by recursively subdividing it into four quadrants or regions Finkel & Bentley (1974), while the three-dimensional analogue is an *octree*. Quadtrees have been used in meshfree methods as background *mesh* generators Klaas & Shephard (2000), Cartwright *et al.* (2001), Griebel & Schweitzer (2002a), Macri *et al.* (2003) and also for adaptivity Tabarraei & Sukumar (2005), The use of binary trees is different in this context. In this thesis binary trees are used not to generate a hierarchical mesh but rather as a tool for building a connectivity map between nodes and Gaussian points.

In a recent work Fasshauer (2007) *binary tree* methods have been applied in meshfree methods, although a different method (Radial Basis Function) is used.

A *kd*-tree (where k stays for $k-$ dimensional tree) is a space-partitioning data structure for organizing N points in a $k-$dimensional space. A complete exposition of *kd-trees* is beyond the aims of this thesis, but main concepts will be briefly recalled. The interested reader can refer to specialized textbooks as Cormen *et al.* (2001) or Moore (1991) for more formal definitions and details. The space-partitioning achieved by hierarchically decomposing a set of points into smaller sets of data, until each subset has the same number of points. The procedure is graphically shown in figure 4.5



(a) The space-partitioning

(b) The final kd-tree

Figure 4.5: Example of kd-tree (from Wikipedia)

The first step consists in picking a point (the *root*) and considering a plane perpendicular to one of the axis and passing through the root. The whole domain is then split in two halves, each one containing roughly the same number of nodes. For each of the two sub-partitions, a node is chosen again, but this time the splitting plane will be on a different dimension of the previous one. The two nodes are called *children*. The procedure is then repeated for the children, generating other children, until the final sub-partitions will not contain any node. These final children are called *leaves*. From the root to the leaves, every node of the tree represents a point. In meshfree methods, the final connectivity matrix is constructed by two steps: the construction of the tree and the *range query*. The computational cost for building the tree is $\mathcal{O}(kN \log N)$ for points in $\mathbb{R}^k$ and the range query takes only $\mathcal{O}(\log N)$, while for the brute force algorithm is $\mathcal{O}(N)$. In Krysl & Belytschko (2001) it is recommended to use $\mathcal{O}(\log N)$ algorithm for the neighbour search.

The *kd*-tree algorithm is considerably more efficient than the naive implementation and it will be preferred in the development of the codes.

4.1.1 Speed-up using a background mesh: a mixed *kd-tree-elements* approach

A further speed-up in neighbour search can be achieved whenever a background element mesh is used for integration purposes. In fact, for this type of integration, Gaussian points do not need to be searched, since they are located within the boundaries of each element. Knowing the inverse mapping between elements and nodes, (i.e. which elements belong to each node), it is necessary to search inside only one set of points, i.e. S_1 (4.4). Since the number of nodes it is usually less than the number of Gaussian points, this approach notably reduces the computational time. However, it can only be used in combination with elements. When a different type of integration is implemented, for example regular background grid, it is necessary to perform the full *kd*-tree neighbour search, which is nevertheless $\mathcal{O}(\log N)$.

4.1.2 The `Matlab` *kd*-tree

`Matlab` does not have a predefined *kd*-tree routine, but a free implementation can be download from the internet at the Matlab Central File Exchange. The one used in this thesis is by Dr. Guy Shechter and is covered by the BSD License.

4.1.3 Results

Numerical tests have been performed to assess the capabilities of the proposed approach. The number of nodes of set S_1 (4.4) has been fixed, while the number of Gaussian points (set S_2 (4.5)) was varied.

As it can be seen in figure 4.6, for small data sets, the brute force algorithm seems to perform better than *kd-tree*. However, this is simply due to overhead in *kd*-tree (the construction of the *tree*). In fact, when the data set grows, the brute force algorithm becomes incredibly slow, even using a highly vectorized version, as in this case. This highly vectorized brute force version needs more storage memory, since it requires the whole *distance* matrix. With the increase of the Gaussian points then, the brute force becomes then prohibitive. In fact for the

case in figure 4.7, with a larger set of nodes, it was not possible to apply the brute force algorithm.

Figures 4.6 and 4.7 show the convenience of using a background element mesh for range searching. Compared to the brute force, the approach described in section 4.1.1 can save up to two orders of magnitude in computational time and an order of magnitude with respect to the *kd*-tree in Gaussian points. Nonetheless, in order to exploit the advantages of brute force for small data sets (e.g. Gaussian points on boundaries for the application of the boundary conditions), the code is able to switch from brute force to *kd*-tree according to the size of the set.

For the Gaussian points in the bulk, the code switches to the *mixed kd*-tree algorithm with nodes and elements.

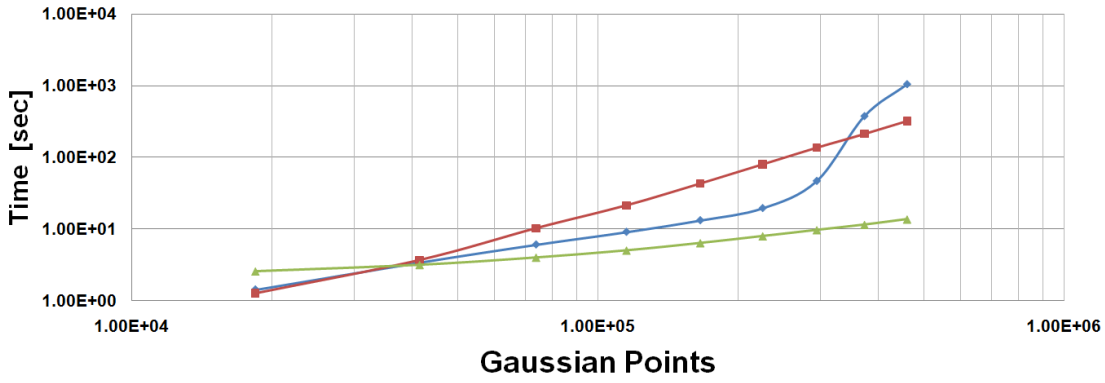


Figure 4.6: Computational times for neighbour search for fixed number of nodes (2465): blue line: brute force; red line: *kd*-tree in Gaussian points; green line: *kd*-tree using background elements

4.2 The Moment Matrix

The key of the reproducing properties of RKPM is the moment matrix $\mathbf{M}(\mathbf{x})$ (3.50).

The formula is here recalled for the sake of clarity

$$\mathbf{M}(\mathbf{x}) = \sum_{I=1}^{n_s} \mathbf{p}\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho}\right)\mathbf{p}^T\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho}\right) w\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho_I}\right), \tag{4.9}$$

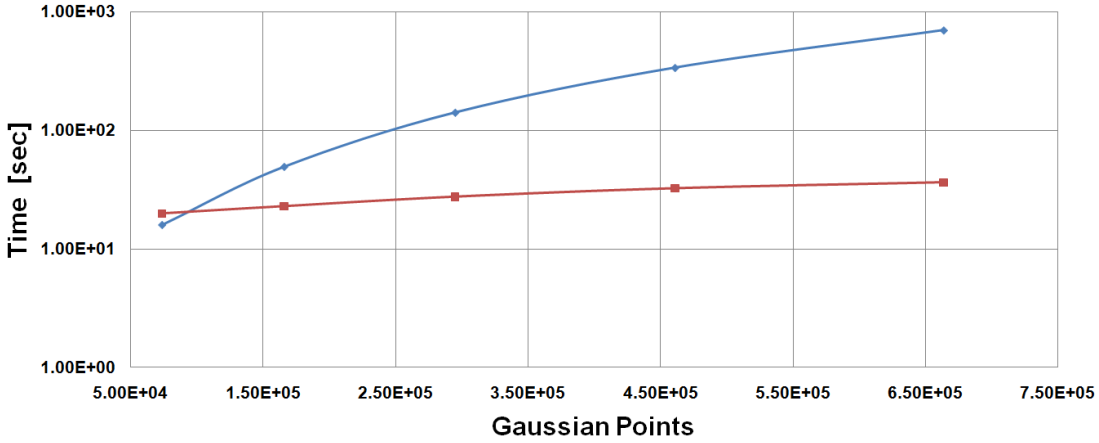


Figure 4.7: Computational times for neighbour search for fixed number of nodes (9537): blue line: *kd*-tree in Gaussian points; red line: *kd*-tree using background elements

where ρ is the mean of all the dilatation parameters and $\mathbf{p}$ is the polynomial basis. For linear reproducing properties in $2D$ is, for example,

$$\mathbf{p}^T\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho}\right) = \begin{bmatrix} 1 & \frac{x-x_I}{\rho} & \frac{y-y_I}{\rho} \end{bmatrix}. \tag{4.10}$$

Once the connectivity vectors are obtained (4.8), the values $\mathbf{s}$ (3.5) can be used to calculate the weight function and its derivatives. Indeed, the connectivity vectors `ig`, `js` and `s` allow the evaluation of the window functions directly on the non-zero values of the compact support functions. With the opportune correction (3.51), these values will become the shape functions. The role of the connectivity vectors is crucial for the code, since they need to be computed only once (with an efficient algorithm $\mathcal{O}(\log N)$) and consent to avoid any loop for the subsequent calculations. All the operations can then be carried out *vector-wise*. This operation is called *loop vectorization*. Moreover, the computations are executed only on non-zero values of the weight functions, avoiding any unnecessary computation. In the next subsections all the operations will be explained in detail.

4.2.1 The window *function handle*

In `Matlab` it is possible to define an object called *function handle*, which allows a quick evaluation of a function. The function handle is defined using the following command:

$$\texttt{Kernel} = \texttt{@(s)}\ \ 1 - 6s^2 + 8s^3 - 3s^4. \tag{4.11}$$

In this case @(s) indicate an *anonymous* variable. Such *mute* variable can be replaced with the actual variable `s` (4.7) and calculate the expression contained in the object `Kernel` over all the values contained in `s` simply by executing the command

$$\texttt{PSI} = \texttt{Kernel(s)}, \tag{4.12}$$

where the vector `PSI`[1] contains the non-zero values of the weight functions for all the nodes and for all the Gaussian points.

Once obtained the values `PSI`, the following operations are necessary to calculate the entries of the moment matrix:

- the computation of the *scaled coordinates* $\xi_I = \frac{x-x_I}{\rho}$ and $\eta_I \frac{y-y_I}{\rho}$,
- the computation of the polynomial basis (4.10),
- the computation of the sum of the polynomial basis (4.9).

4.2.2 Computation of scaled coordinates

Supposing `GRIDSET` is the coordinate list of the set S_1 and `GGRID` the coordinate list of the set S_2 and `CSI` is the computer implementation of $\xi_I\ \forall I = 1, \ldots, n_s$ and `ETA` the computer variable for $\eta_I\ \forall I = 1, \ldots, n_s$, then the computation of ξ_I and η_I is done in the following way:

$$\texttt{CSI} = (\texttt{GGRID(ig, 1)} - \texttt{GRIDSET(js, 1)})./\texttt{mrho} \tag{4.13}$$

$$\texttt{ETA} = (\texttt{GGRID(ig, 2)} - \texttt{GRIDSET(js, 2)})./\texttt{mrho} \tag{4.14}$$

where `mrho` is ρ the average of the dilatation parameters.

[1] `PSI` is used instead of `W` to avoid confusion with the Gaussian weights

4.2.3 Computation of the polynomial basis

Similarly to the command (4.11), it is possible to define the polynomial basis as *function handle*. For a more neat programming style, it is opportune to group these functions in an array, or *cell* array.

A *cell array* is similar to a multidimensional array, but with the interesting feature that data are allowed to be not necessarily of the same type.

```
PolyBasis.p{1} = @(x,y) 1                    (4.15)
PolyBasis.p{2} = @(x,y) x                    (4.16)
PolyBasis.p{3} = @(x,y) y                    (4.17)
```

```
PolyBasis.dpdx{1} = @(x,y) 0                 (4.18)
PolyBasis.dpdx{2} = @(x,y) 1                 (4.19)
PolyBasis.dpdx{3} = @(x,y) 0                 (4.20)
```

```
PolyBasis.dpdy{1} = @(x,y) 0                 (4.21)
PolyBasis.dpdy{2} = @(x,y) 0                 (4.22)
PolyBasis.dpdy{3} = @(x,y) 1                 (4.23)
```

Let us define a function handle of the type

```
P_handle = @(P) {P(CSI,ETA)}                 (4.24)
```

where `CSI` is (4.13) and `ETA` is (4.14). In this way, by simply calling

```
P = cellfun(P_handle,PolyBasis.p(1:3)).'     (4.25)
```

the values of the polynomial basis are readily calculated. Moreover, with the call

```
DPDX = cellfun(P_handle,PolyBasis.dpdx(1:3)).'     (4.26)
```

```
DPDY = cellfun(P_handle,PolyBasis.dpdy(1:3)).'
```
(4.27)

the derivatives of the polynomial basis can also be automatically calculated, without the need of defining dedicated function handles for the derivatives.

4.2.4 Computation of the sum of the polynomials

According to equation (4.9), in order to obtain the entries of the moment matrix, the sum of the polynomials (4.25) must be performed over the nodes. After (4.25), `P` is a 3×1 cell array. Every element of the cell contains n_{nz} values, the same number as the variables `ig`, `js` and `s`.

The sum is carried out in three steps:

1. the product

$$\mathbf{p}_i\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho_I}\right)\mathbf{p}_j\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho_I}\right)w\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho_I}\right) \qquad i,j=1,\ldots,n_p, \tag{4.28}$$

   where n_p is the number of the polynomials included in the basis;

2. reshape of the matrix product (4.28);

3. sum over the columns (the second dimension) of the reshaped matrix.

The matrix product (4.28) originates n_p^2 vectors of size $n_{nz} \times 1$. However, since the moments matrix is symmetric, not all the products need to be stored, but only $n_p(n_p+1)/2$. Therefore in equation (4.28) the products to calculate are only the ones $i,j=1,\ldots,n_p$ with $i \leq j$.

These *indexes* i,j for a symmetric matrix $n_p \times n_p$ can be grouped in two arrays of length $n_p(n_p+1)/2$. For example, if $n_p = 3$

$$\mathtt{i} = \begin{bmatrix}1 & 1 & 1 & 2 & 2 & 3\end{bmatrix} \tag{4.29}$$

$$\mathtt{j} = \begin{bmatrix}1 & 2 & 3 & 2 & 3 & 3\end{bmatrix} \tag{4.30}$$

Indexes in (4.29) and (4.30) allow the definition of a *function handle* for the products in equation (4.28)

```
M_handle=@(i,j) {P{i}.*P{j}.*PSI}
```
(4.31)

In fact, vectors `P`$_i$ and `PSI` are the same size and thus can be multiplied in an *element-wise* manner. The call

```
M = arrayfun(M_handle,i,j)
```
(4.32)

originates a cell array of size $n_p(n_p + 1)/2$, where each element contains the product (4.28).

The next step is to reshape each element of the cell array `M` from a n_{nz} vector to a matrix $n_g \times n_s$. Luckily, the command `sparse` in `Matlab` allows this transformation without filling with zeros, but keeping the indexes `ig`, `js`.

The command

```
S = sparse(i,j,s,m,n)
```
(4.33)

uses vectors `i`, `j`, and `s` to generate an $m \times n$ sparse matrix such that

$$S(i(k), j(k)) = s(k) \qquad k = 1, \ldots, n_{nz} \tag{4.34}$$

if vectors `i`, `j`, and `s` are all the same length.

Additionally, any elements of `s` that are zero are ignored, along with the corresponding values of `i` and `j`. Any elements of `s` that have duplicate values of `i` and `j` are added together.

Therefore, the following command can be defined

```
spar_handle = @(C) sum(sparse(ig,js,C,ng,ns),2)
```
(4.35)

where `ng` is n_g and `ns` is n_s.

Command (4.35) incorporates 2 commands, the first one is `sparse`, which reshape the *anonymous* matrix `C` and the second one is the `sum` over the nodes, where 2 stays for the second dimension of the matrix, i.e. the columns. It can be seen that with the introduction of the connectivity vectors `ig` and `js`, it is no longer necessary to search for nodes at each Gaussian point for the purpose of the sum in the moment matrix. These operations can be indeed automatized once the connectivity has been defined at the beginning of the calculation.

By calling

```
M = cellfun(spar_handle,M)
```
(4.36)

the moment matrix $\mathbf{M}$ is finally calculated. The result of (4.36) is a cell array of $n_p(n_p+1)/2$ elements, each array containing n_g elements, i.e. as many as the number of the Gaussian points. By inverting $\mathbf{M}$, it is possible to get the corrective terms for the kernels (the weight functions) that restore the reproducing properties.

4.2.5 Derivatives of the moment matrix

For completeness are here reported the commands necessary to calculate the derivatives of the moment matrix. Firstly, it is necessary to calculate the derivatives of the kernel, as in equation (3.9).

$$\frac{\partial w}{\partial x} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \xi_I}\frac{\partial \xi_I}{\partial x} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \xi_I}\frac{1}{\rho_I} \tag{4.37}$$

$$\frac{\partial w}{\partial y} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \eta_I}\frac{\partial \xi_I}{\partial x} = \frac{\partial w}{\partial s}\frac{\partial s}{\partial \eta_I}\frac{1}{\rho_I} \tag{4.38}$$

The first step is calculating $\dfrac{\partial w}{\partial s}$ by defining the following function handle

$$\texttt{dKernel = @(s)} \; -12s(s-1)^2 \tag{4.39}$$

and then executing the command

$$\texttt{dPSI = dKernel(S).} \tag{4.40}$$

The derivatives $\dfrac{\partial s}{\partial \xi_I}$ and $\dfrac{\partial s}{\partial \eta_I}$ are calculated as follows

```
dDcsi= ((GGRID(ig,1)-GRIDSET(js,1))./rho(js))./S        (4.41)
                                    dDcsi(S==0) = 1     (4.42)

dDeta= ((GGRID(ig,2)-GRIDSET(js,2))./rho(js))./S        (4.43)
                                    dDeta(S==0) = 1     (4.44)
```

where equations (4.42) and (4.44) eliminate the indeterminate form as described in section (3.1.2). Derivatives of the kernel functions are

```
dPSIcsi = dPSI.*dDcsi                                    (4.45)
dPSIeta = dPSI.*dDeta                                    (4.46)
```

The handles for the derivatives of the moment matrix are

```
DMx_handle = @(i,j) {(DPDX{i}.*P{j} + P{i}.*DPDX{j}).*PSI
                     + P{i}.*P{j}.*dPSIcsi}              (4.47)

DMy_handle = @(i,j) {(DPDY{i}.*P{j} + P{i}.*DPDY{j}).*PSI
                                    + P{i}.*P{j}.*dPSIeta}  (4.48)
```

and the operations of product, reshape and sum are executed by calling these commands

```
DMx = arrayfun(DMx_handle,i,j)                           (4.49)
DMx = cellfun(spar_handle,DMx)                           (4.50)
DMy = arrayfun(DMy_handle,i,j)                           (4.51)
DMy = cellfun(spar_handle,DMy)                           (4.52)
```

4.2.6 Explicit Inverse of the Moments Matrix

In section 4.2 the construction if the moment matrix for linear reproducing property was illustrated.

If a linear reproducing property is desired, the moments matrix can be inverted directly by symbolic manipulation, as in Breitkopf *et al.* (2000) and Zhou *et al.* (2005) . This can still be done in 3D, when the moment matrix is size 4×4.

However, if higher order reproducing properties are sought, this approach cannot be used, since the symbolic inversion of a matrix of size larger than 5×5 is unstable or even impossible to achieve practically. In section 4.3.1 an alternative approach is proposed. It will be shown that the explicit inverse of the moments

matrix for linear reproducing property is indeed the starting point to calculate the inverse of the moment matrix with higher reproducing capabilities.

The inversion of the moment matrix for linear reproducing properties can be done following the Cramer's rule

$$\det(\mathbf{M}) = (\mathbf{M}_{2,2}\mathbf{M}_{3,3} - \mathbf{M}_{2,3}^2)\mathbf{M}_{1,1} - \mathbf{M}_{1,3}^2\mathbf{M}_{2,2} - \mathbf{M}_{1,2}^2\mathbf{M}_{3,3} + \\ + 2\mathbf{M}_{1,2}\mathbf{M}_{1,3}\mathbf{M}_{2,3} \quad (4.53)$$

$$\mathbf{M}_{1,1}^{-1} = \frac{\mathbf{M}_{2,2}\mathbf{M}_{3,3} - \mathbf{M}_{2,3}^2}{\det(\mathbf{M})} \quad (4.54)$$

$$\mathbf{M}_{1,2}^{-1} = -\frac{\mathbf{M}_{1,2}\mathbf{M}_{3,3} - \mathbf{M}_{1,3}\mathbf{M}_{2,3}}{\det(\mathbf{M})} \quad (4.55)$$

$$\mathbf{M}_{1,3}^{-1} = \frac{\mathbf{M}_{1,2}\mathbf{M}_{2,3} - \mathbf{M}_{1,3}\mathbf{M}_{2,2}}{\det(\mathbf{M})} \quad (4.56)$$

$$\mathbf{M}_{2,2}^{-1} = \frac{\mathbf{M}_{1,1}\mathbf{M}_{3,3} - \mathbf{M}_{1,3}^2}{\det(\mathbf{M})} \quad (4.57)$$

$$\mathbf{M}_{2,3}^{-1} = -\frac{\mathbf{M}_{1,1}\mathbf{M}_{2,3} - \mathbf{M}_{1,2}\mathbf{M}_{1,3}}{\det(\mathbf{M})} \quad (4.58)$$

$$\mathbf{M}_{3,3}^{-1} = \frac{\mathbf{M}_{1,1}\mathbf{M}_{2,2} - \mathbf{M}_{1,2}^2}{\det(\mathbf{M})} \quad (4.59)$$

Derivatives of the inverse of the moments matrix can be calculated with equation (3.55).

4.3 Inversion of the Moment Matrix through Partitioning

In the previous section, explicit inversion formulas for the moments matrix were illustrated, for the case of linear reproducibility. This section is focused on the following two questions:

1. how to calculate the shape functions if a higher order reproducibility is sought?

2. how to calculate the shape functions if other points are inserted?

The first question is usually known as *h-adaptivity*, whilst the second is known *p-adaptivity*. Together, they are known as *hp*-adaptivity. The *hp*-adaptivity consists in a global/local refinement of the approximation through raising the order of the polynomials in the approximating function and/or adding nodes to the discretization, with the aim of reducing the local error or capturing high gradient zones. Local h-refinement consists in adding nodes to limited zones of the domain, usually detected when a *measure* of the error exceeds a predefined threshold, while global *h-refinement* means refining the discretization in the whole domain. Local p-refinement is done by enriching the approximation with the introduction of extra-unknowns, as explained in section 3.8.2 and they not need to be necessarily polynomials. Global p-refinement is instead obtained in meshfree methods by adding extra-functions to the polynomial basis in equation (3.42). This section will deal on global p-refinement, while enrichments will be treated in the next chapters to simulate delamination.

With the aim of reducing the error, the *error* must be calculated. Of course, if the *error* was known *a priori*, there would be no need of solving the equation in the first place; therefore what it is usually done is *estimating the error a posteriori*. Detailed informations on adaptivity in meshless methods can be found in Liu & Tu (2002a) and Rabczuk & Belytschko (2005). Once these zones are individuated, then nodes or polynomials must be practically inserted into the approximation. In meshfree methods, this task is simpler than FE, as stated in section 1.1. This section explains how practically insert extra polynomials or nodes to the shape functions without carrying out expensive operations. Practical computer implementations can be found in section 5.2.3. It will be shown that the *classic* algorithm in figure 4.1 will be modified into the one in figure 4.8

4.3.1 Fast Inversion of the Moment Matrix and *p*-adaptivity

In this section is proposed a method to answer to the first of the questions in 4.3, i.e. to efficiently invert the moment matrix whenever the order of reproducibility is greater than one. As a result, it can be proficiently used for global p-adaptivity. A similar approach based on the imposing consistency constraints can be found in Breitkopf *et al.* (2000). The reproducing properties are imposed

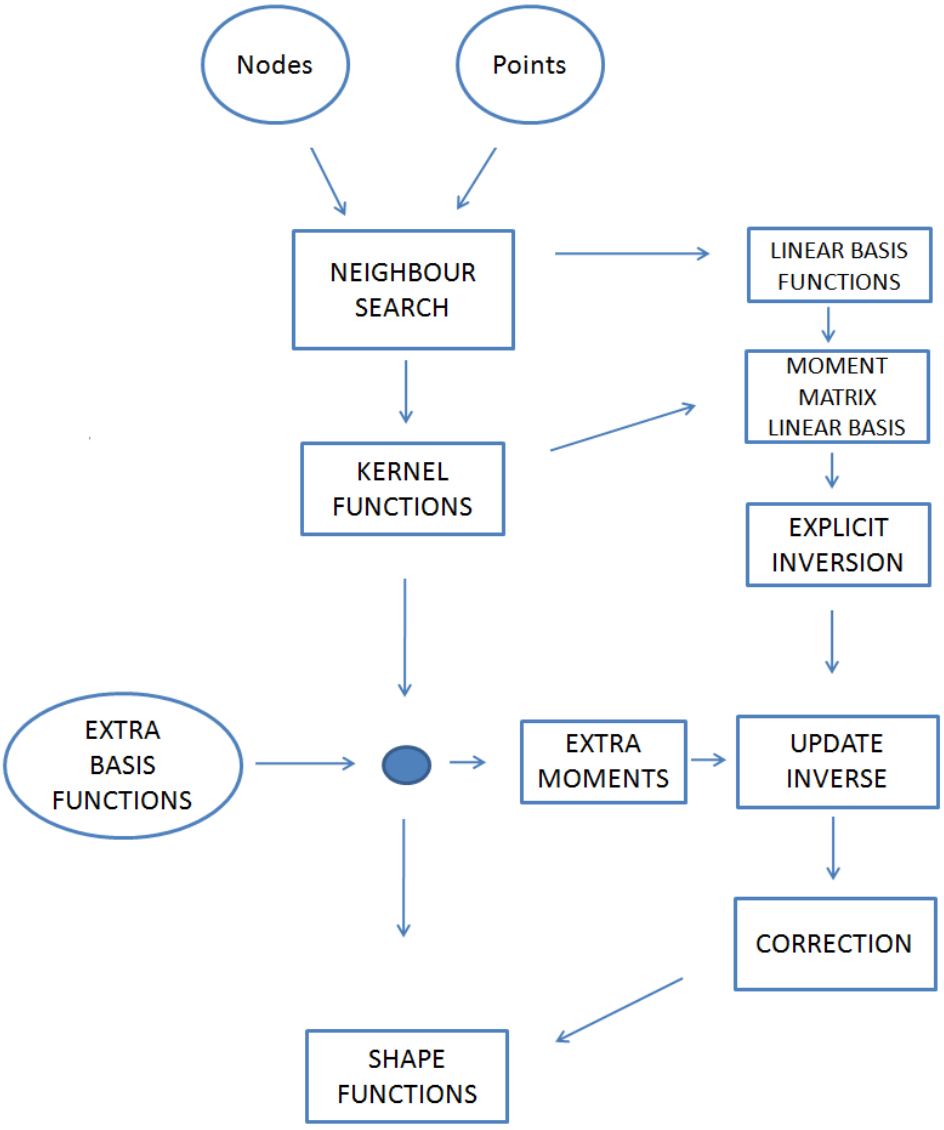


Figure 4.8: Flowchart of the proposed algorithm in meshfree methods

through Lagrangian multipliers. The resulting method does not make use of a moment matrix, but iteratively update the weights, nonetheless leading to similar formulas. The proposed method does not employ Lagrangian multipliers but simply a partitioning of the moment matrix and it is based on the updating of its inverse. The codes described in the next chapters are entirely based on this approach, instead of numerical routines (Gauss elimination, LU factorization) for point-wise inversion of the moments matrix.

Let

$$w_I = w\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho_I}\right) \quad I = 1,\ldots,N \tag{4.60}$$

and

$$\mathbf{p}_I^{(k)} = \mathbf{p}^{(k)}\left(\frac{\mathbf{x}-\mathbf{x}_I}{\rho}\right), \tag{4.61}$$

where ρ is

$$\rho = \frac{1}{N}\sum_{I=1}^{N}\rho_I. \tag{4.62}$$

The superscript k in equation (4.61) is the number of polynomial terms included in the basis functions $\mathbf{p}$. In fact, shape functions with such basis will be indicated with superscript k as well, i.e.

$$\phi_I^{(k)}(\mathbf{x}) = \mathbf{p}^{(k)}(0)^T\mathbf{M}_k^{-1}(\mathbf{x})\mathbf{p}_I^{(k)}w_I(\mathbf{x}), \tag{4.63}$$

$$\mathbf{M}_k(\mathbf{x}) = \sum_{I=1}^{N}\mathbf{p}_I^{(k)}\mathbf{p}_I^{(k)T}w_I, \tag{4.64}$$

where the *scaled* form has been used to avoid problems on the condition number for $\mathbf{M}_k(\mathbf{x})$. The final goal is to obtain the shape functions when an additional term is included in the basis function, without re-calculating the moment matrix and, most importantly, without performing the costly *point-by-point* matrix inversion.

This means

$$\phi_I^{(k+1)}(\mathbf{x}) = \mathbf{p}^{(k+1)}(0)^T\mathbf{M}_{k+1}^{-1}(\mathbf{x})\mathbf{p}_I^{(k+1)}w_I(\mathbf{x}). \tag{4.65}$$

In order to do so, the *corrective* term needs to be updated

$$C_I^{(k+1)}(\mathbf{x}) = \mathbf{p}^{(k+1)}(0)^T\mathbf{M}_{k+1}^{-1}(\mathbf{x})\mathbf{p}_I^{(k+1)}, \tag{4.66}$$

which in turn requires the update of the basis

$$\mathbf{p}_I^{(k+1)T} = \begin{bmatrix} \mathbf{p}_I^{(k)T} & p_I^{(e)} \end{bmatrix} \tag{4.67}$$

and the update of the inverse of the moment matrix

$$\mathbf{M}_{k+1}(\mathbf{x}) = \sum_{I=1}^{N} \mathbf{p}_I^{(k+1)} \mathbf{p}_I^{(k+1)T} w_I = \sum_{I=1}^{N} \begin{bmatrix} \mathbf{p}_I^{(k)} \\ p_I^{(e)} \end{bmatrix} \begin{bmatrix} \mathbf{p}_I^{(k)T} & p_I^{(e)} \end{bmatrix} w_I = \\ = \begin{bmatrix} \mathbf{M}_k(\mathbf{x}) & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix}, \tag{4.68}$$

where

$$\mathbf{b}(\mathbf{x}) = \sum_{I=1}^{N} \mathbf{p}^{(k)} p^{(e)} w_I \tag{4.69}$$

$$c(\mathbf{x}) = \sum_{I=1}^{N} p^{(e)2} w_I. \tag{4.70}$$

Inversion of matrix (4.68) can be directly obtained from the entries of $\mathbf{M}_{k+1}^{-1}(\mathbf{x})$ without performing a computationally expensive *point by point* inversion. In fact, using Boltz's formula, $\mathbf{M}_{k+1}(\mathbf{x})$ can be inverted *block-wise* by using the following analytical inversion formula

$$\mathbf{M}_{k+1}^{-1}(\mathbf{x}) = \begin{bmatrix} \mathbf{M}_k^{-1}(\mathbf{x}) & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{q} \begin{bmatrix} (\mathbf{M}_k^{-1}\mathbf{b})(\mathbf{M}_k^{-1}\mathbf{b})^T & -\mathbf{M}_k^{-1}\mathbf{b} \\ -(\mathbf{M}_k^{-1}\mathbf{b})^T & 1 \end{bmatrix}, \tag{4.71}$$

where

$$q(\mathbf{x}) = c - \mathbf{b}^T(\mathbf{M}_k^{-1}\mathbf{b}). \tag{4.72}$$

Equation (4.71) is tremendously useful since one need to perform only *element-wise* products and divisions that are particularly fast in `Matlab`.

In fact, without this formula, one has to *loop* over the far numerous Gaussian points and invert the moment matrix with a numerical routine. This is one of the major drawbacks of RKPM and MLS, since an accurate integration of the weak form of the PDE might necessitate a high order Gaussian quadrature scheme and therefore the number of Gaussian points notably grows. Instead, using equation (4.71), assuming that the entries $\mathbf{M}_k^{-1}(\mathbf{x})$ are already available, one needs only

to calculate the moments $\mathbf{b}$ and c and perform only *element-wise* operations to update the moment matrix.

Boltz' formula allows to *vectorize* the `Matlab` code and effectively speeds up the calculation of the shape functions.

It must be pointed out though, that equation (4.71) shows how to *update* the inverse of a matrix when an additional column (and a row) is introduced when $\mathbf{M}_k^{-1}(\mathbf{x})$ is already known. Therefore to fully exploit the advantages of the block-inversion, the entries of $\mathbf{M}_k^{-1}(\mathbf{x})$ must be obtained beforehand. Nevertheless, one of the greatest advantage of equation (4.71) is that can be applied *iteratively*, in the sense that can be applied several times until the desired order of reproduction is achieved.

For example, a linear reproduction property is at least required for the convergence of the weak form. Therefore shape functions at least must satisfy linear reproduction conditions, which lead to a 3×3 moment matrix in two dimensions and a 4×4 moment matrix in three-dimensions. These sizes of matrices are easily invertible analytically, for example with Cramer's rule for the two-dimensional case and with symbolic inversion for 4×4, that again do not require any numerical routine for matrix inversion and can be obtained with element-wise matrix operations.

It must be remarked that symbolic inversion is *unstable* (and sometimes impossible for some symbolic softwares) for 5×5 and *impossible* to obtain for matrices of higher size. Nonetheless linear reproduction property can be the *starting point* of the inversion of a bigger size matrix. In fact once $\mathbf{M}_3^{-1}(\mathbf{x})$ is calculated for the two-dimensional case, with (4.71) it is possible to obtain $\mathbf{M}_4^{-1}(\mathbf{x})$. Once obtained $\mathbf{M}_4^{-1}(\mathbf{x})$ one proceeds to $\mathbf{M}_5^{-1}(\mathbf{x})$ and so on until the sought order of reproduction is achieved.

4.3.1.1 First Order Derivatives

In the assembly of the stiffness matrix of a weak form, at least first order derivatives of the shape functions are required, hence a fast computation of the first

order derivatives of equations (4.65) can accelerate the assembly process.

$$\frac{\partial \phi_I}{\partial x}^{(k+1)} = \frac{\partial C_I}{\partial x}^{(k+1)} w_I + C_I^{(k+1)} \frac{\partial w_I}{\partial x}, \tag{4.73}$$

where

$$\frac{\partial C_I}{\partial x}^{(k+1)} = \mathbf{p}^{(k)T}(0) \left(\frac{\partial \mathbf{M}_{k+1}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{p}_I^{(k+1)} + \mathbf{M}_{k+1}^{-1}(\mathbf{x}) \frac{\partial \mathbf{p}_I^{(k+1)}}{\partial \mathbf{x}} \right). \tag{4.74}$$

Derivatives of $\mathbf{M}_{k+1}^{-1}(\mathbf{x})$ can be readily obtained by derivation of equation (4.71)

$$\begin{aligned} \frac{\partial \mathbf{M}_{k+1}^{-1}(\mathbf{x})}{\partial x} = & \begin{bmatrix} \dfrac{\partial \mathbf{M}_k^{-1}(\mathbf{x})}{\partial x} & 0 \\ 0 & 0 \end{bmatrix} - \frac{1}{q^2} \frac{\partial q}{\partial x} \begin{bmatrix} (\mathbf{M}_k^{-1}\mathbf{b})(\mathbf{M}_k^{-1}\mathbf{b})^T & -\mathbf{M}_k^{-1}\mathbf{b} \\ -(\mathbf{M}_k^{-1}\mathbf{b})^T & 1 \end{bmatrix} + \\ & + \frac{1}{q} \begin{bmatrix} \dfrac{\partial (\mathbf{M}_k^{-1}\mathbf{b})}{\partial x} (\mathbf{M}_k^{-1}\mathbf{b})^T + (\mathbf{M}_k^{-1}\mathbf{b}) \dfrac{\partial (\mathbf{M}_k^{-1}\mathbf{b})^T}{\partial x} & -\dfrac{\partial (\mathbf{M}_k^{-1}\mathbf{b})}{\partial x} \\ -\dfrac{\partial (\mathbf{M}_k^{-1}\mathbf{b})^T}{\partial x} & 0 \end{bmatrix}, \end{aligned} \tag{4.75}$$

where

$$\frac{\partial \mathbf{M}_k^{-1}(\mathbf{x})}{\partial \mathbf{x}} = -\mathbf{M}_k^{-1}(\mathbf{x}) \frac{\partial \mathbf{M}_k(\mathbf{x})}{\partial x} \mathbf{M}_k^{-1}(\mathbf{x}), \tag{4.76}$$

$$\frac{\partial (\mathbf{M}_k^{-1}\mathbf{b})}{\partial x} = \frac{\partial \mathbf{M}_k^{-1}(\mathbf{x})}{\partial x} \mathbf{b} + \mathbf{M}_k^{-1}(\mathbf{x}) \frac{\partial \mathbf{b}}{\partial x}, \tag{4.77}$$

$$\frac{\partial \mathbf{M}_k(\mathbf{x})}{\partial x} = \sum_{I=1}^{N} \left(\frac{\partial \mathbf{p}^{(k)}}{\partial x} \mathbf{p}^{(k)T} + \mathbf{p}^{(k)} \frac{\partial \mathbf{p}^{(k)T}}{\partial x} \right) w_I + \mathbf{p}^{(k)} \mathbf{p}^{(k)T} \frac{\partial w_I}{\partial x}, \tag{4.78}$$

$$\frac{\partial q}{\partial x} = \frac{\partial c}{\partial x} - \frac{\partial \mathbf{b}^T}{\partial x} (\mathbf{M}_k^{-1}\mathbf{b}) + \mathbf{b}^T \frac{\partial (\mathbf{M}_k^{-1}\mathbf{b})}{\partial x}, \tag{4.79}$$

$$\frac{\partial c}{\partial x} = \sum_{I=1}^{N} 2p^{(e)} \frac{\partial p^{(e)}}{\partial x} w_I + p^{(e)2} \frac{\partial w_I}{\partial x}, \tag{4.80}$$

$$\frac{\partial \mathbf{b}}{\partial x} = \sum_{I=1}^{N} \left(\frac{\partial \mathbf{p}^{(k)}}{\partial x} p^{(e)} + \mathbf{p}^{(k)} \frac{\partial p^{(e)}}{\partial x} \right) w_I + \mathbf{p}^{(k)} p^{(e)} \frac{\partial w_I}{\partial x}. \tag{4.81}$$

4.3.2 Fast Inversion of the Moment Matrix and *h*-adaptivity

In this section is proposed a method to answer to the second of the questions in 4.3.

Let us indicate with subscript N the moment matrix with N nodes

$$\mathbf{M}_N(\mathbf{x}) = \sum_{I=1}^{N} \mathbf{p}_I \mathbf{p}_I^T w_I. \tag{4.82}$$

If an additional node is introduced, the moment matrix inverse needs to be updated.

$$\mathbf{M}_{N+1}(\mathbf{x}) = \sum_{I=1}^{N+1} \mathbf{p}_I \mathbf{p}_I^T w_I = \sum_{I=1}^{N} \mathbf{p}_I \mathbf{p}_I^T w_I + \mathbf{p}_{N+1}\mathbf{p}_{N+1}^T w_{N+1} = \mathbf{M}_N(\mathbf{x}) + \\ + \mathbf{p}_{N+1}\mathbf{p}_{N+1}^T w_{N+1}. \tag{4.83}$$

Therefore it is simply a *rank-1* update:

$$\mathbf{M}_{N+1}^{-1}(\mathbf{x}) = \mathbf{M}_N^{-1}(\mathbf{x}) - \frac{(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})^T}{r} w_{N+1}, \tag{4.84}$$

$$r = 1 + \mathbf{p}_{N+1}^T (\mathbf{M}_N^{-1}\mathbf{p}_{N+1}) w_{N+1}, \tag{4.85}$$

therefore

$$\phi_I(\mathbf{x}) = \mathbf{p}^T(0)\mathbf{M}_{N+1}^{-1}(\mathbf{x})\mathbf{p}_I w_I(\mathbf{x}) \qquad I = 1, \ldots, N+1. \tag{4.86}$$

Derivatives are given by

$$\frac{\partial \phi_I}{\partial x} = \mathbf{p}^T(0) \left[\left(\frac{\partial \mathbf{M}_{N+1}^{-1}(\mathbf{x})}{\partial x} \mathbf{p}_I + \mathbf{M}_{N+1}^{-1}(\mathbf{x}) \frac{\partial \mathbf{p}_I}{\partial x} \right) w_I(\mathbf{x}) + \right. \\ \left. + \mathbf{M}_{N+1}^{-1}(\mathbf{x}) \mathbf{p}_I \frac{\partial w_I}{\partial x} \right], \tag{4.87}$$

$$\frac{\partial \mathbf{M}_{N+1}^{-1}(\mathbf{x})}{\partial x} = \frac{\partial \mathbf{M}_N^{-1}(\mathbf{x})}{\partial x} - \frac{1}{r^2} \left[\left(\frac{\partial (\mathbf{M}_N^{-1}\mathbf{p}_{N+1})}{\partial x} (\mathbf{M}_N^{-1}\mathbf{p}_{N+1})^T + \right. \right. \\ \left. \left. + (\mathbf{M}_N^{-1}\mathbf{p}_{N+1}) \frac{\partial (\mathbf{M}_N^{-1}\mathbf{p}_{N+1})^T}{\partial x} \right) r - (\mathbf{M}_N^{-1}\mathbf{p}_{N+1})(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})^T \frac{\partial r}{\partial x} \right] w_{N+1} - \\ + \frac{(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})^T}{r} \frac{\partial w_{N+1}}{\partial x}, \tag{4.88}$$

where

$$\frac{\partial(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})}{\partial x} = \frac{\partial \mathbf{M}_N^{-1}(\mathbf{x})}{\partial x}\mathbf{p}_{N+1} + \mathbf{M}_N^{-1}(\mathbf{x})\frac{\partial \mathbf{p}_{N+1}}{\partial x}, \tag{4.89}$$

$$\frac{\partial r}{\partial x} = \left[\frac{\partial \mathbf{p}_{N+1}^T}{\partial x}(\mathbf{M}_N^{-1}\mathbf{p}_{N+1}) + \mathbf{p}_{N+1}^T\frac{\partial(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})}{\partial x}\right] w_{N+1} + \\ + \mathbf{p}_{N+1}^T(\mathbf{M}_N^{-1}\mathbf{p}_{N+1})\frac{\partial w_{N+1}}{\partial x}. \tag{4.90}$$

Finally, it should be remarked that the approach in formula (4.83) was firstly noted in You *et al.* (2003), without explicitly suggesting the update of the *inverse* of the moment matrix.

4.4 The Kernel Corrective Term

The term in equation (3.52) (here recalled for clarity) is called corrective because it allows to calculate the shape functions (4.91) from the kernel functions, restoring the reproducing properties.

$$\phi_I(\mathbf{x}) = C_I(\mathbf{x})w\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right), \tag{4.91}$$

$$C_I(\mathbf{x}) = \mathbf{p}^T(\mathbf{0})\mathbf{M}(\mathbf{x})^{-1}\mathbf{p}^T\left(\frac{\mathbf{x}_I - \mathbf{x}}{\rho}\right). \tag{4.92}$$

The values of the weight functions are computed through the commands (4.12) and stored in an array `PSI` of length n_{nz} and so do the values of `P`, stored in a cell array. The problem is the entries of the inverse of the moment matrix $\mathbf{M}^{-1}$, that are stored in a cell array `Minv` and every element of the cell is an array of length n_g. Therefore, a *naive* approach would loop over the basis function, then double loop over the nodes and over the evaluation points, calculating the correction and then applying it to the weight functions. Instead, the corrective term can be quickly calculated using the connectivity vector `ig`.

The code is the following:

```
C = 0;
if nargout>2
```

```
    dCx = 0;
    dCy = 0;
end

for i=1:np
    for j=1:np
       if P0{i}~=0
            if i<=j
                C = C + P0{i}.*Minv{i,j}(ig).*P{j};
                if nargout>2
                    dCx = dCx + P0{i}.*DMinvx{i,j}(ig).*P{j} +...
                    + P0{i}.*Minv{i,j}(ig).*DPDX{j};
                    dCy = dCy + P0{i}.*DMinvy{i,j}(ig).*P{j} +...
                    P0{i}.*Minv{i,j}(ig).*DPDY{j};
                end
            else
                C = C + P0{i}.*Minv{j,i}(ig).*P{j};
                if nargout>2
                    dCx = dCx + P0{i}.*DMinvx{j,i}(ig).*P{j} +...
                    + P0{i}.*Minv{i,j}(ig).*DPDX{j};
                    dCy = dCy + P0{i}.*DMinvy{j,i}(ig).*P{j} +...
                    + P0{i}.*Minv{j,i}(ig).*DPDY{j};
                end
            end
       end
    end
end
```

where `ig` act as a *index* in the command `Minv{i,j}(ig)`. The variable `nargout` is the number of output arguments. If more than two arguments are requested, derivatives are calculated.

This index `ig` points to `Minv{i,j}(:)`, in the same order `PSI` and `P` are stored, allowing the generic entry of the moment matrix to become an array of length n_{nz} and multiply `PSI` and `P` in an *element-wise* manner. The same approach can

be applied also to the derivatives of the corrective term, i.e. `dCx` and `dCy`. Using the same logic, the following handle can be defined

```
asb_handle = @(f,g) f.*g(ig)
```
(4.93)

that will be useful for the stiffness matrix assembly in the next section. Finally, the shape functions are calculated as follows

```
PHI = C.*PSI;
if nargout>2
    DPHIx = (dCx.*PSI + C.*dPSIcsi)/mrho;
    DPHIy = (dCy.*PSI + C.*dPSIeta)/mrho;
end
```

At this stage, though, the values are stored in arrays of size n_{nz}. To reshape the arrays, the following handle can be defined

```
sp_handle = @(f) sparse(ig,js,f,ng,ns).
```
(4.94)

The object (4.94) transforms a sparse array f defined by indexes `ig` and `js` into a sparse matrix `ng` $\times$ `ns`.

The inverse operation can be performed by the following command

```
desp_handle = @(f) f(sub2ind(size(f),ig,js)).
```
(4.95)

The final step for constructing the shape functions is

```
PHI = sp_handle(PHI);
if nargout>2
     DPHIx = sp_handle(DPHIx);
     DPHIy = sp_handle(DPHIy);
end
```

4.5 Assembly of the Stiffness Matrix

In this section the assembly of the stiffness matrix in equation (3.103) is explained in detail for the two and three dimensional cases.

The infinitesimal strain is defined as

$$\epsilon = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial}{\partial x} & \\ & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \mathcal{L}\mathbf{u}, \tag{4.96}$$

where $\mathcal{L}$ is the infinitesimal strain differential operator and $\mathbf{u}$ is the displacement vector.

The linear elastic stress-strain relationship is defined through the Generalized Hooke's law (with the *Voigt notation*[1])

$$\sigma = \mathbf{C}\epsilon = \begin{bmatrix} C_{11} & C_{12} & \\ C_{12} & C_{22} & \\ & & C_{33} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ 2\gamma_{xy} \end{bmatrix}, \tag{4.97}$$

where $\mathbf{C}$ is called the *stiffness tensor* and for two-dimensional orthotropic materials in *plane stress* tensional state is defined as

$$\mathbf{C} = \begin{bmatrix} C_{11} & C_{12} & \\ C_{12} & C_{22} & \\ & & C_{33} \end{bmatrix} = \begin{bmatrix} E_{11}/(1-\nu_{12}\nu_{21}) & \nu_{12}E_{22}/((1-\nu_{12}\nu_{21}) & \\ \nu_{12}E_{22}/((1-\nu_{12}\nu_{21}) & E_{22}/(1-\nu_{12}\nu_{21}) & \\ & & G_{12} \end{bmatrix}, \tag{4.98}$$

where E_{ii} is the Young's modulus along axis i, G_{ij} is the shear modulus in direction j on the plane whose normal is in direction i, ν_{ij} is the Poisson's ratio that corresponds to a contraction in direction j when an extension is applied in direction i.

If *plane strain* is desired, then the following changes are necessary

[1] The Voigt notation is the standard mapping for tensor indexes

$$E_{11} \leftarrow \frac{E_{11}}{1-\nu_{12}^2}, \tag{4.99}$$

$$E_{22} \leftarrow \frac{E_{22}}{1-\nu_{12}^2}, \tag{4.100}$$

$$G_{12} \leftarrow \frac{G_{12}}{1-\nu_{12}^2}, \tag{4.101}$$

$$\nu_{12} \leftarrow \frac{\nu_{12}}{1-\nu_{12}^2}, \tag{4.102}$$

$$\nu_{21} \leftarrow \frac{\nu_{21}}{1-\nu_{12}^2}. \tag{4.103}$$

$$\tag{4.104}$$

The displacement vector $\mathbf{u}$ is approximated with the meshfree shape functions

$$\mathbf{u}(\mathbf{x}) \approx \mathbf{u}^h(\mathbf{x}) = \begin{bmatrix} \underbrace{\mathbf{\Phi}^T(\mathbf{x})}_{1\times n_s} & \\ & \underbrace{\mathbf{\Phi}^T(\mathbf{x})}_{1\times n_s} \end{bmatrix} \begin{bmatrix} \underbrace{\mathbf{U}}_{n_s\times 1} \\ \underbrace{\mathbf{V}}_{n_s\times 1} \end{bmatrix}. \tag{4.105}$$

It is opportune to distinguish between the mathematical vector $\mathbf{\Phi}(\mathbf{x})$ which is size $n_s \times 1$ and the computer array `PHI` in section 4.4, which is a sparse matrix of size $n_g \times n_s$.

Applying (4.96) to (4.105), the strain resulting from the approximation is

$$\epsilon^h(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial \mathbf{\Phi}^T}{\partial x} & \\ & \dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ \dfrac{\partial \mathbf{\Phi}^T}{\partial y} & \dfrac{\partial \mathbf{\Phi}^T}{\partial x} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \end{bmatrix} \tag{4.106}$$

and applying (4.97) to (4.106)

$$\sigma^h = \begin{bmatrix} C_{11} & C_{12} & \\ C_{12} & C_{22} & \\ & & C_{33} \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{\Phi}^T}{\partial x} & \\ & \dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ \dfrac{\partial \mathbf{\Phi}^T}{\partial y} & \dfrac{\partial \mathbf{\Phi}^T}{\partial x} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \end{bmatrix} = \begin{bmatrix} C_{11}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} + C_{12}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ C_{11}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} + C_{22}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ C_{33}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} + C_{33}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \end{bmatrix}. \tag{4.107}$$

In equation (3.99) the variational term for the stiffness matrix can be expressed as follows

$$\int_\Omega \delta\epsilon^T \sigma d\Omega = \begin{bmatrix} \delta\mathbf{U}^T & \delta\mathbf{V}^T \end{bmatrix} \int_\Omega \begin{bmatrix} \dfrac{\partial \mathbf{\Phi}}{\partial x} & & \dfrac{\partial \mathbf{\Phi}}{\partial y} \\ & \dfrac{\partial \mathbf{\Phi}}{\partial y} & \dfrac{\partial \mathbf{\Phi}}{\partial x} \end{bmatrix} \begin{bmatrix} C_{11}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} + C_{12}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ C_{11}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} + C_{22}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} \\ C_{33}\dfrac{\partial \mathbf{\Phi}^T}{\partial y} + C_{33}\dfrac{\partial \mathbf{\Phi}^T}{\partial x} \end{bmatrix} \mathrm{d}\Omega \begin{bmatrix} \mathbf{U} \\ \mathbf{V} \end{bmatrix}. \tag{4.108}$$

Therefore the stiffness matrix $\mathbf{K}$ can be written as

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{12}^T & \mathbf{K}_{22} \end{bmatrix}, \tag{4.109}$$

where

$$\mathbf{K}_{11} = \int_\Omega C_{11}\frac{\partial \mathbf{\Phi}}{\partial x}\frac{\partial \mathbf{\Phi}^T}{\partial x} + C_{33}\frac{\partial \mathbf{\Phi}}{\partial y}\frac{\partial \mathbf{\Phi}^T}{\partial y}\mathrm{d}\Omega, \tag{4.110}$$

$$\mathbf{K}_{22} = \int_\Omega C_{33}\frac{\partial \mathbf{\Phi}}{\partial x}\frac{\partial \mathbf{\Phi}^T}{\partial x} + C_{22}\frac{\partial \mathbf{\Phi}}{\partial y}\frac{\partial \mathbf{\Phi}^T}{\partial y}\mathrm{d}\Omega, \tag{4.111}$$

$$\mathbf{K}_{12} = \int_\Omega C_{12}\left(\frac{\partial \mathbf{\Phi}}{\partial x}\frac{\partial \mathbf{\Phi}^T}{\partial y} + \left(\frac{\partial \mathbf{\Phi}}{\partial x}\frac{\partial \mathbf{\Phi^T}}{\partial y}\right)^T\right)\mathrm{d}\Omega. \tag{4.112}$$

The stiffness matrix can be then computed whenever the following matrices

are calculated

$$\mathbf{K}_{xx} = \int_{\Omega} \frac{\partial \mathbf{\Phi}}{\partial x} \frac{\partial \mathbf{\Phi}^T}{\partial x} \mathrm{d}\Omega, \tag{4.113}$$

$$\mathbf{K}_{yy} = \int_{\Omega} \frac{\partial \mathbf{\Phi}}{\partial y} \frac{\partial \mathbf{\Phi}^T}{\partial y} \mathrm{d}\Omega, \tag{4.114}$$

$$\mathbf{K}_{xy} = \int_{\Omega} \frac{\partial \mathbf{\Phi}}{\partial x} \frac{\partial \mathbf{\Phi}^T}{\partial y} \mathrm{d}\Omega. \tag{4.115}$$

Integrals in (4.113), (4.114) and (4.115) are calculated using Gaussian quadrature. Naming `W` as the vector of size $n_g \times 1$ containing the Gaussian weights associated with the Gaussian points `GGRID`, then a brute force algorithm would be

```
for i=1:ns
    for j=1:ns
       Kxx(i,j) = W.'*(DPHIx(:,i).*DPHIy(:,j))
    end
end
```

Double loops however, really slow down the calculation. To avoid one loop, one could then think to create an intermediate matrix

```
for i=1:nset
     WX(i,:) = DPHIx(:,i)'.*W.';
end
Kxx = WX*DPHIx;
```

To avoid loops, one could then think to calculating an intermediate matrix

```
WX = DPHIx.*repmat(W,1,ns)
```

and then

```
Kxx = WX.'*DPHIy
```

The command `repmat` creates n_s replicas of the vector `W` along the columns. Nevertheless, such an approach is costly in term of speed and memory storage, since it creates a temporary array of size $n_g \times n_s$.

Instead, *indexing* can be used with the handle in equation (4.93). With the command `desp_handle` (4.95)

```
DPHIx = desp_handle(DPHIx)
```
(4.116)

the derivatives of the shape functions are reshaped into mono-dimensional arrays.

```
WX = asb_handle(DPHIx,W)
```
(4.117)

Then reshaping `DPHIx` and `WX` into bi-dimensional arrays with (4.94)

```
DPHIx = sp_handle(DPHIx)
```
(4.118)

```
WX = sp_handle(WX)
```
(4.119)

Ffinally equation (4.113) is computed

```
KXX = WX.'*DPHIx
```
(4.120)

The same approach can be repeated for equations (4.115) and (4.114).

4.6 Results

In order to show the effectiveness of the improvements, in this section it will be presented a comparison between the *classical* and the *new* methods based on the computational run-times.

By *classic* it is intended the method based on a *point-wise* inversion of the moment matrix (4.9), i.e. the moment matrix is calculated and then the inversion is made by looping over the Gaussian points, create a temporary matrix, invert it with a standard numerical routine (LU factorization or Gaussian elimination) and then pass it back to the cell array `Minv`. Instead, the *new* method is the one explained in section 4.3 whose computer implementation is explained in detail in appendix 5.2.3. For both methods, the routines used for the connectivity, the correction and the assembly are the same, i.e. the procedures explained respectively in sections 4.1, 4.4 and 4.5.

To assure a fair comparison, both methods have been executed on the same computer, with a 3 GHz processor Intel ®Pentium ®with 1 GB of RAM. Nevertheless, performances may vary and computational times reduce on different computers. Hence, the computational times reported here should be intended as a relative measure, not as an absolute measure of the code performances.

As a benchmark case, a three-dimensional model of a bar has been considered as in figure 4.9, with elastic properties and geometry resumed in respectively in tables 4.1 and 4.2.

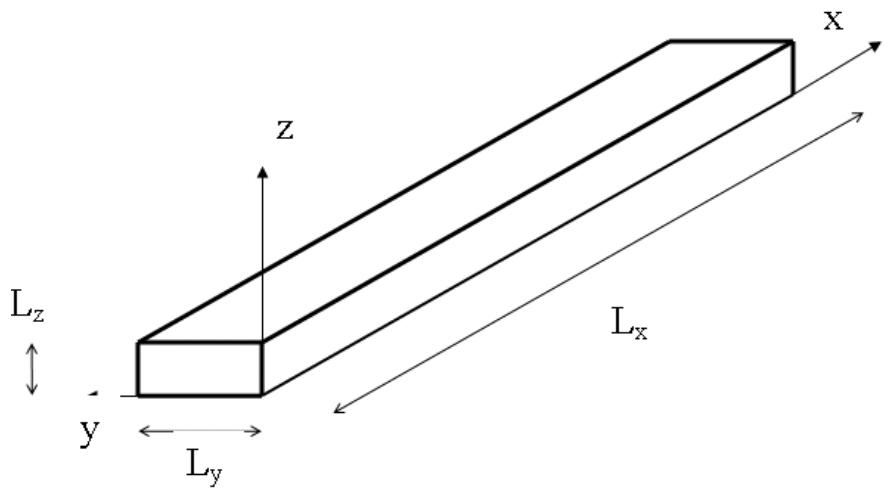


Figure 4.9: Three dimensional case: geometry

E_{11}	$E_{22} = E_{33}$	G_{12}	G_{23}	$\nu_{12} = \nu_{13}$	ν_{23}
122.7 GPa	10.1 GPa	5.5 GPa	3.7 GPa	0.25	0.45

Table 4.1: Elastic Properties for the benchmark case

L_x	L_y	L_z
185 mm	25 mm	2.5 mm

Table 4.2: Geometry for the benchmark case

The quadrature cells considered are cubic as in figure 4.10

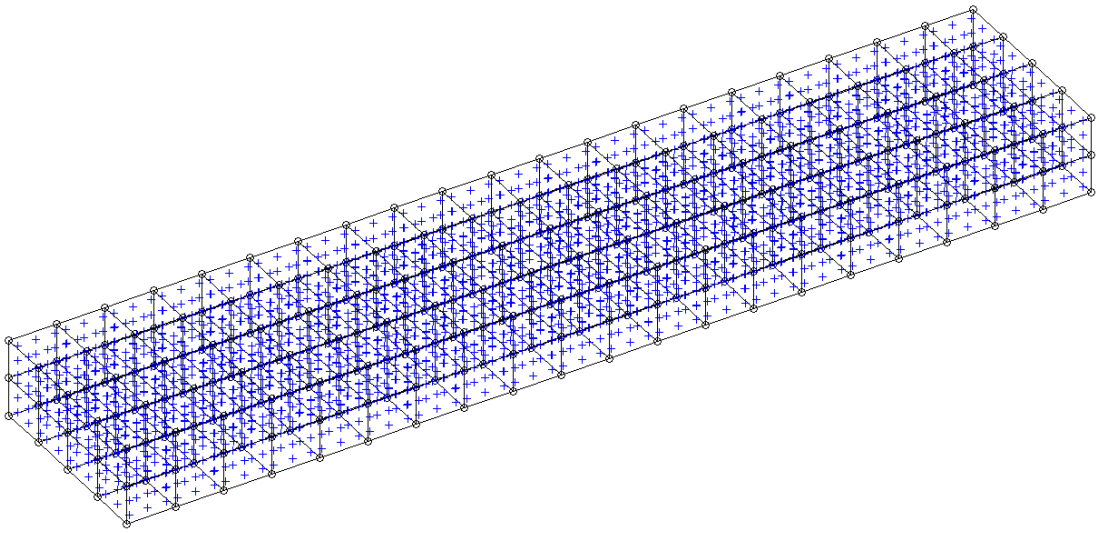


Figure 4.10: Continuous Line: Quadrature Cells; Circles: Nodes; Blue crosses: Gaussian Points

and a full quadratic basis (superscript 2) has been included as in equation (4.121), thus the moment matrix has size 10×10.

$$\mathbf{p}^{(2)T}(\mathbf{x}) = \begin{bmatrix} 1 & x & y & z & x^2 & y^2 & z^2 & xy & yz & xz \end{bmatrix}. \tag{4.121}$$

Tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 illustrates the computational run-times for the specimen for same number of nodes ($11 \times 6 \times 3 = 132$), but different number of Gaussian Points. The dilatation parameter ρ_I is variable with the node and it is chosen as 2.4 $max(h_I)$ with h_I the maximum length of the edges concurring in that node.

The run-time has been divided mainly in time for the creation of the *shape functions* and time for the *assembly* of the stiffness matrix, as described in subsection 4.5. The run-times for the shape functions comprise also the derivatives of the shape functions. The run-time for the shape functions has been subdivided in 4 contributions:

1. *connectivity*, i.e. the time necessary to calculate the vectors `ig`, `js` and `s` using the *kd*-tree algorithm as described in subsection 4.1;

2. *moment matrix*, i.e. the time necessary to compute the cell array `M` as in subsection 4.2;

3. *inversion of the moment matrix*, i.e. the time for inverting the moment matrix;

4. *correction term*, i.e. the time for calculating the correction to the kernel as in subsection 4.4.

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*
800	0.1614	0.1269
2700	0.4329	0.4500
6400	0.8698	0.8728
12500	1.5980	1.5694
21600	3.0833	3.2427
34300	4.8564	4.2063
51200	7.4106	7.2022

Table 4.3: Computational run-times for the connectivity

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*
800	0.2220	0.3374
2700	2.6361	2.5686
6400	4.7568	4.7648
12500	7.1372	7.7983
21600	12.5379	18.1758
34300	25.3300	24.7946
51200	34.6201	33.2414

Table 4.4: Computational run-times for the correction term

It can be observed that, as expected, the major differences are not in the connectivity (table 4.3), in the correction term (table 4.4) and in the assembly (table 4.5). In fact, for these purposes, the same routines have been used in

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*
800	0.3424	0.3908
2700	1.0952	1.0500
6400	2.9279	3.1260
12500	7.2566	5.9309
21600	14.7860	11.1589
34300	22.0235	21.5802
51200	29.3857	31.1248

Table 4.5: Computational run-times for the assembly

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*
800	1.8273	**0.5012**
2700	11.4439	**1.8473**
6400	22.4772	**3.9266**
12500	49.4992	**8.1483**
21600	75.7696	**14.1422**
34300	135.0388	**21.9535**
51200	170.9730	**35.3794**

Table 4.6: Computational run-times for the moment matrix

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*
800	1.5319	**1.8981**
2700	12.6961	**9.5511**
6400	58.8198	**21.1465**
12500	190.0229	**39.9969**
21600	555.5533	**72.0766**
34300	1344.9519	**117.3779**
51200	3002.2513	**162.7680**

Table 4.7: Computational run-times for the inversion of the moment matrix

Gaussian Points	**Classic** *(sec)*	**New** *(sec)*	**New/Classic** *%*
800	3.7471	**2.8693**	**76.57 %**
2700	27.2389	**14.4471**	**53.04 %**
6400	86.9840	**30.7691**	**35.37 %**
12500	248.3761	**57.6299**	**23.20 %**
21600	647.1185	**107.8531**	**16.67 %**
34300	1510.5029	**168.6082**	**11.16 %**
51200	3215.7002	**239.0169**	**7.43 %**

Table 4.8: Computational Run-times for the Shape Functions

both methods. Instead, there is a great difference for the computation times related to the moment matrix (table 4.6), even though the same procedure of subsection 4.2 has been used for both cases. The reason for such discrepancy is that in the *classic* method, a symmetric 10×10 moment matrix need to be constructed straight after the connectivity phase, whereas with the new method (the partitioning inversion method) only a symmetric 4×4 moment matrix is necessary. The symmetric 4×4 moment matrix for the partitioning method can be inverted symbolically and represents the starting point of the iterative procedure described in subsection 4.3. The remaining terms in the new method are constructed at the same time of the inversion, allowing a huge saving in terms of computational time, around 80% average for all the cases.

This saving is not only in terms of speed but also in terms of storage memory. In fact, the number of entries of the moment matrix stored in the classic method is $10 \times 11/2 = 55$, whereas in the partitioning method is $4 \times 5/2 = 10$. Therefore, an estimation of the saving in terms of speed is $(55 - 10)/55 \times 100 = 81.18\%$.

Moreover, for the storage memory, there is another aspect to consider, which is not evident from the computational times. After the inversion, the memory occupied by the moment matrix (4×4) can be cleared since it is no longer necessary. The update is executed directly on the *inverse* of the moment matrix, through previous calculation of the moments $\mathbf{b}$ (equation (4.69) and c (equation (4.70)). After the update is executed, the moments $\mathbf{b}$ and c required for the update of the inverse can be deleted as well. On the contrary, for the classic method, all the

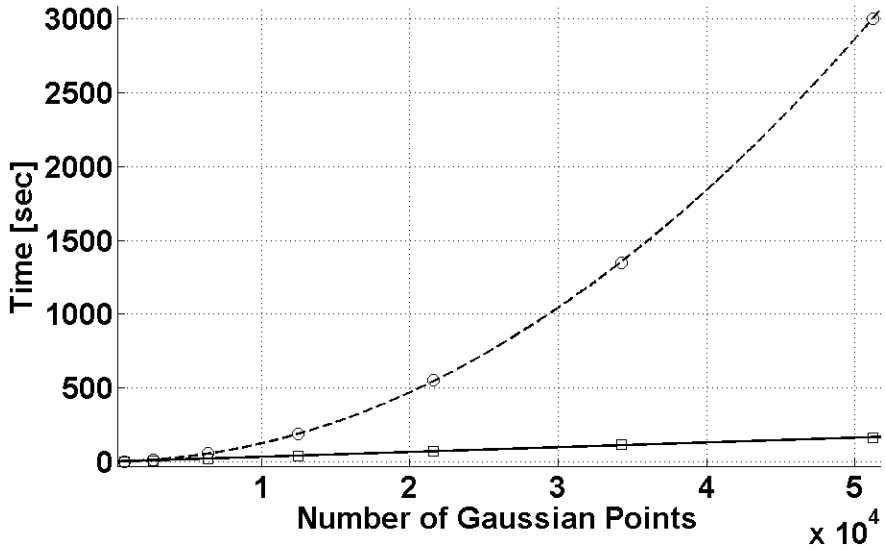


Figure 4.11: Computational run-times for the inversion of the moment matrix: circles: classic method; squares: new method; dashed line: quadratic fit for the classic method; continuous line: linear fit for the new method

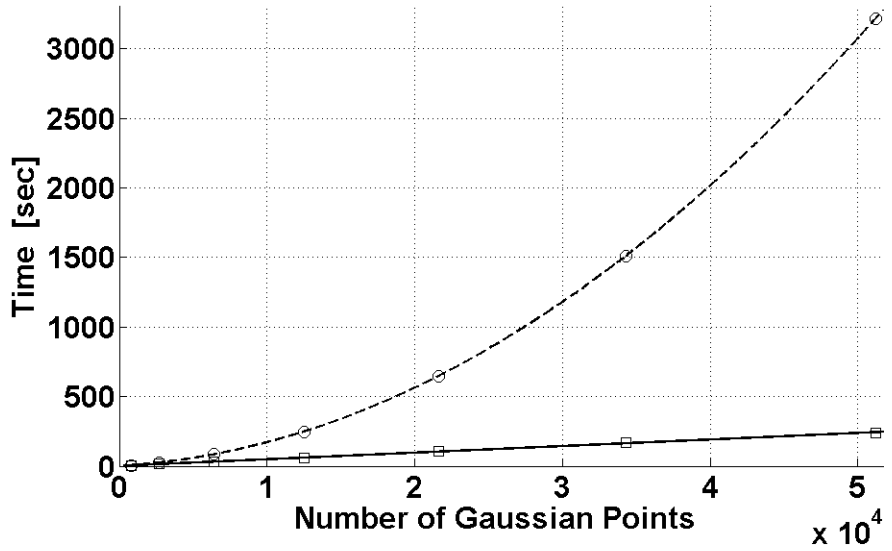


Figure 4.12: Computational run-times for the shape functions: circles: classic method; squares: new method; dashed line: quadratic fit for the classic method; continuous line: linear fit for the new method

Coefficients	95 % Coefficient Bounds
$p_1 = 389.7$	$(377.6, 401.8)$
$p_2 = 789.8$	$(777.2, 802.4)$
$p_3 = 403.9$	$(390.5, 417.3)$
Coefficient of determination R^2	0.999

Table 4.9: Inversion of the moment matrix (classic method): goodness of quadratic fit $f(x) = p_1x^2 + p_2x + p_3$

Coefficients	95 % Coefficient Bounds
$p_1 = 60.24$	$(56.9, 63.59)$
$p_2 = 60.69$	$(57.59, 63.78)$
Coefficient of determination R^2	0.9977

Table 4.10: Inversion of the moment matrix (new method) : goodness of linear fit $f(x) = p_1x + p_2$

terms must be retained for the *point-wise* inversion, and deleted only after the inversion is concluded. The inversion through partitioning of the moment matrix is particularly advantageous, as shown in table 4.7. Moreover, figures 4.11 and 4.12 show another interesting side: the dataset of the computational times and the number of Gaussian points can be fit by a quadratic curve for the classic method and by a *linear* curve for the proposed method.

The classic method is fit by a quadratic polynomial with coefficients and statistics resumed in table 4.9, while the ones for the proposed method are in table 4.10. In both cases, the 95% coefficients bounds are pretty tight with a coefficient of determination very close to 1.

The proposed method, either for the inversion of the moment matrix and for the construction of the shape functions, has a computational cost $\mathcal{O}(n)$ with respect to the number of Gaussian Points, compared to the $\mathcal{O}(n^2)$ of the classic one. The saving in computational time is tremendous: in fact, for the classic method (table 4.8), for 51200 Gaussian points, almost an hour (54 minutes) is needed to calculate the shape functions and their first derivatives, while with the proposed method only 4 minutes.

Coefficients	95 % Coefficient Bounds
$p_1 = 383.7$	$(380.5, 386.9)$
$p_2 = 874$	$(870.7, 877.3)$
$p_3 = 491$	$(487.5, 494.6)$
Coefficient of determination R^2	0.999

Table 4.11: Shape functions (classic method) : goodness of quadratic fit $f(x) = p_1x^2 + p_2x + p_3$

Coefficients	95 % Coefficient Bounds
$p_1 = 88.01$	$(83.85, 92.16)$
$p_2 = 88.74$	$(84.9, 92.59)$
Coefficient of determination R^2	0.9983

Table 4.12: Shape functions (new method) : goodness of linear fit $f(x) = p_1x + p_2$

Chapter 5

Description of the Codes

In this chapter the prototype codes in `Matlab` will be presented. Firstly, the purposes of the Object-Oriented Programming (OOP) will be briefly illustrated in section 5.1, although a complete and rigorous description of the ideas can be found in specialized textbooks. In this chapter the main ideas of the OOP will be used to create the classes for the two-dimensional and three-dimensional codes.

5.1 The object-oriented programming

The most widely known programming technique is the *procedural* programming. In the procedural coding, the data are variables or structures and the instructions are executed in a sequential order, by calling *functions* that have these data as inputs and/or outputs.

These functions perform one or more operations on the data. After these operations are performed, the transformed data are passed back to the main program which will continue to execute the list of the instructions.

The OOP is a rather different approach. In the OOP the main variables are *objects*. An object is a specific instance of a *class*. A class is, from a programming point of view, a complex structure which encapsulates both variables and functions. The variables of the class are called *properties* and the functions defined on these variables are called *methods*. The methods are not executed sequentially, but only if a specific action is desired to be carried out on the class.

When the input variables are given to the class, the class becomes an object. The creation of an object, in this sense, is performed by a method called *constructor* which is compulsorily defined for every class. The constructor has the same name of the class.

The class, in fact, describes the characteristics that all the (different) objects have in common. Analogously to the procedural programming, the objects are the variables, while the class is similar to the list of the instructions. The difference is that the list of the instructions does not have to be executed obligatorily and the variables are internal to the object. Object can then manage their own data and different object can interact according to the actions (the *methods*) defined on them.

Another interesting feature of the classes is that they can alert when any property value is queried or changed. These notices are called *events* and objects can broadcast the when for example a property value is changed. As a consequence, other methods called *listeners* can be defined. Listeners execute other functions in response to the notification of an event[1].

5.2 The class RKPM

The class RKPM applies to both two-dimensional and three-dimensional linear elastic objects. The purpose of this class is to provide the RKPM approximation whenever evaluation points are inserted as input. This class is usually called to compute the shape functions in the Gaussian points, either for the domain integral in the stiffness matrix and for the line integrals for the force vectors and for the constraints.

5.2.1 The properties

The main properties defined for the class RKPM are

- nodes: vector $n_s \times 2$ or $n_s \times 3$; a list of the coordinates of the RKPM nodes (4.4): initially it is set as the vertices of the integration mesh, but other nodes can be added to the list if h-adaptivity is performed

[1]The classes presented in this chapter do not implement listeners and events at the moment

- `rho`: dilatation parameters as in section 3.1.1
- `PSI` ,`dPSIcsi`, `dPSIeta` : the values of the kernel function and its derivatives stored in a *vector-wise* manner, as defined in equations (4.12), (4.45) and (4.46)
- `ig`, `js`: the connectivity vectors as defined in (4.8)
- `PolyBasis`: cell vector of function handles; contains the polynomial basis functions as in equation (4.15).
- `Kernel` `dKernel`: function handles defining the functional form of the kernel and its first derivative, as in equations (4.11) and (4.11)
- `CSI` and `ETA`: the scaled and translated coordinates as in equations (4.13) and (4.14)
- `P` ,`P0`, `DPDX`, `DPDY` cell-arrays containing the values of the polynomial basis as defined in equations (4.25), (4.26) and (4.27) stored in a *vector-wise* manner
- `Minv`, `DMinvx`,`DMinvy` cell-arrays containing the values of the inverse of the moment matrix and their derivatives, as it will be described in section 5.2.3
- `sp_handle` : the function handle described in 4.94
- `asb_handle`: the function handle described in 4.93
- `desp_handle` : the function handle described in 4.95
- `PHI`, `DPHIx`, `DPHIy` the values of the shape functions and their derivatives as described in 4.4, stored in a sparse manner
- `pe`, `dpedx`, `dpedy` cell arrays of function handles containing additional polynomials to be added to the basis, as in p-adaptivity: in fact, as shown later, the linear polynomial basis is inserted by *default* in `PolyBasis`

5.2.2 The methods

The methods defined on the class RKPM are:

1. RKPM(varargin) : the *constructor*, named as the class itself, allows the construction of the object. The input variables are the function handles that define the kernel, its derivative and the optional enhanced polynomials.

```
function obj = RKPM(varargin)
 obj.nodes = @(a)(a.fv.vertices);
 obj.rho = @(a)(a.rho);
 obj.Kernel = varargin{1};
 obj.dKernel = varargin{2};
 obj.PolyBasis.p{1} = inline('sparse(1:numel(x),1,1)','x','y');
 obj.PolyBasis.p{2} = inline('x','x','y');
 obj.PolyBasis.p{3} = inline('y','x','y');
 obj.PolyBasis.dpdx{1} = inline('sparse(numel(x),1)','x','y');
 obj.PolyBasis.dpdx{2} = inline('sparse(1:numel(x),1,1)','x','y');
 obj.PolyBasis.dpdx{3} = inline('sparse(numel(x),1)','x','y');
 obj.PolyBasis.dpdy{1} = inline('sparse(numel(x),1)','x','y');
 obj.PolyBasis.dpdy{2} = inline('sparse(numel(x),1)','x','y');
 obj.PolyBasis.dpdy{3} = inline('sparse(1:numel(x),1,1)','x','y');
 obj.nPB = 3;
 obj.nPE = numel(varargin{3});
 obj.pe = varargin{3};
 obj.dpedx = varargin{4};
 obj.dpedy = varargin{5};
end
```

2. ShapeAndDer: computes the shape functions and their derivatives in the input points GGRID. This method firstly calls the function Conn that implements the *kd-tree* (section 4.1) and subsequently calls the kernel handles.

```
function obj = ShapeAndDer(obj,GGRID,obj2)
```

```
 [obj.PSI,obj.ig,obj.js,obj.dPSIcsi,obj.dPSIeta] =...
  Conn(obj.nodes(obj2),GGRID,obj.rho(obj2),...
  ...obj.Kernel,obj.dKernel);

 if ~isempty(obj.ig)
[obj,obj.PHI,obj.DPHIx,obj.DPHIy] =...
RKPM3(obj.nodes(obj2),...
GGRID,obj,mean(obj.rho(obj2)),...
...obj.pe,obj.dpedx,obj.dpedy);
end

obj.CSI = []; obj.ETA = [];
end
```

Once obtained the values for `PSI`, `ig`, `js`, `dPSIcsi` and `dPSIeta`, the function `RKPM3` actually calculates the shape functions.

It is usually employed for the assembly of the stiffness matrix.

3. `Shape`: same as `ShapeAndDer`, but only calculates the shape functions, not the derivatives. It is usually employed for the assembly of the force vectors in equation.

4. `ShapeAndDerPU`: same as `ShapeAndDer` but calculates only the Partition of Unity functions. It is used to construct the enrichment as described later in section 6.4.

5. `ShapePU`: same as `Shape` but calculates only the Partition of Unity functions

6. `ShapeWithPsi`, `ShapeAndDerWithPsi`, `ShapeWithPsiPU` and `ShapeAndDerWithPsiPU` are similar to the above methods, with the exception that the connectivity and the kernel values are provided as inputs, because calculated elsewhere, for example by the following method `GetFromPlate`. These methods are usually used for the enrichments.

7. GetFromPlate a method that *selects* the ig, js PSI and optionally dPSIcsi and dPSIeta from an existing set of nodes-Gaussian points.

```
function obj = GetFromPlate(obj,obj2,nodes)
     [c,loc] = ismember(obj2.js,nodes);
     obj.ig = obj2.ig(c);
     obj.js = loc(c);
     obj.PSI = obj2.PSI(c);
     if ~isempty(obj2.dPSIcsi)
           obj.dPSIcsi = obj2.dPSIcsi(c);
           obj.dPSIeta = obj2.dPSIeta(c);
     end
end
```

It should be noted that, even though Minv is defined as a property, there is no property for the moment matrix. In fact, the moment matrix entries are calculated *internally* in the function RKPM3.

5.2.3 Inversion of the moments matrix through partitioning

The function RKPM3[1] is defined internally in the class RKPM and its purpose is to calculate the shape functions in the nodes and their derivatives.

The operations in RKPM3 have been already described in chapter 4, with the exception of the details regarding the computer implementation of the inversion of the moments matrix.

The inversion of the moments matrix is carried out by another function RKPMp3[2], which is called (iteratively) after the symbolic inversion of the moments matrix with linear reproduction properties, as described in subsection 4.2.6.

The inputs to RKPMp3 are the object obj, the properties pe, dpedx and dpedy and the function handle spar_handle (4.35), used for the computation of the moments.

[1] the number 3 refers to the third version of the function

[2] p stands for partitioning or p-adaptivity

Firstly, using CSI and ETA, the properties P ,P0, DPDX and DPDY are augmented by the $i-th$ function in pe, dpedx and dpedy

```
np = numel(obj.PolyBasis.p);
obj.P0{np+1,1} = pe(0,0);
obj.P{np+1,1} = pe(obj.CSI,obj.ETA);
obj.PolyBasis.p{1,np+1} = pe;
if nargin>4
   obj.DPDX{np+1,1} = dpedx(obj.CSI,obj.ETA);
   obj.DPDY{np+1,1} = dpedy(obj.CSI,obj.ETA);
   obj.PolyBasis.dpdx{1,np+1} = dpedx;
   obj.PolyBasis.dpdy{1,np+1} = dpedy;
end
```

The **b** vector in equation (4.69) (and its derivatives (4.81)) is calculated using the spar_handle

```
b_handle = @(i) {obj.P{i}.*obj.P{np+1}.*obj.PSI};
b = arrayfun(b_handle,1:np);
b = cellfun(spar_handle,b);
if nargin>3
  dbdx_handle = @(i) {obj.DPDX{i}.*obj.P{np+1}.*obj.PSI +
  + obj.P{i}.*obj.DPDX{np+1}.*obj.PSI +
   + obj.P{i}.*obj.P{np+1}.*obj.dPSIcsi};
  dbdx = arrayfun(dbdx_handle,(1:np).');
  dbdx = cellfun(spar_handle,dbdx);
  dbdy_handle = @(i) {obj.DPDY{i}.*obj.P{np+1}.*obj.PSI +
  + obj.P{i}.*obj.DPDY{np+1}.*obj.PSI +
  +obj.P{i}.*obj.P{np+1}.*obj.dPSIeta};
  dbdy = arrayfun(dbdy_handle,(1:np).');
  dbdy = cellfun(spar_handle,dbdy);
end
```

Same process for c in equation (4.70) and its derivatives (4.80)

```
c = obj.P{np+1}.^2.*obj.PSI;
```

```
c = cell2mat(cellfun(spar_handle,{c}));
if nargin>3
     dcdx = 2.*obj.P{np+1}.*obj.DPDX{np+1}.*obj.PSI +
     + obj.P{np+1}.^2.*obj.dPSIcsi;
     dcdx = cell2mat(cellfun(spar_handle,{dcdx}));
     dcdy = 2.*obj.P{np+1}.*obj.DPDY{np+1}.*obj.PSI +
     +  obj.P{np+1}.^2.*obj.dPSIeta;
     dcdy =  cell2mat(cellfun(spar_handle,{dcdy}));
end
```

The product $\mathbf{M}_k^{-1}\mathbf{b}$ (and its derivatives) is calculated by looping over the number of polynomials

```
for i=1:np
 for j=1:np
    if i<=j
       Minvb{i,1} = Minvb{i,1} + obj.Minv{i,j}.*b{j};
       if nargin>3
          dMinvbdx{i,1} = dMinvbdx{i,1} + obj.DMinvx{i,j}.*b{j} +
           + obj.Minv{i,j}.*dbdx{j};
           dMinvbdy{i,1} = dMinvbdy{i,1} + obj.DMinvy{i,j}.*b{j} +
           +obj.Minv{i,j}.*dbdy{j};
        else
           Minvb{i,1} = Minvb{i,1} + obj.Minv{j,i}.*b{j};
        if nargin>3
          dMinvbdx{i,1} = dMinvbdx{i,1} + obj.DMinvx{j,i}.*b{j} +
             + obj.Minv{j,i}.*dbdx{j};
           dMinvbdy{i,1} = dMinvbdy{i,1} + obj.DMinvy{j,i}.*b{j} +
             + obj.Minv{j,i}.*dbdy{j};
        end
       end
    end
 end
end
```

The q term in equation (4.72) and its derivatives (4.79) are calculated as:

```
for i=1:np
   bTMinvb = bTMinvb + b{i}.*Minvb{i};
   if nargin>4
       dbTMinvbdx = dbTMinvbdx + dbdx{i}.*Minvb{i} +
       + b{i}.*dMinvbdx{i};
       dbTMinvbdy = dbTMinvbdy + dbdy{i}.*Minvb{i} +
       + b{i}.*dMinvbdy{i};
   end
end
q = c-bTMinvb;
if nargin>4
      dqdx = dcdx - dbTMinvbdx;
      dqdy = dcdy - dbTMinvbdy;
end
```

Finally, the update of the inverse of the moment matrix (4.71) (and its derivatives 4.75) is done by

```
for i=1:np+1
  for j=1:np+1
    if i<=j
       if i<=np && j<=np
         obj.Minv{i,j} = obj.Minv{i,j} + Minvb{i}.*Minvb{j}./q;
           if nargin>4
           obj.DMinvx{i,j} = obj.DMinvx{i,j} +
           - dqdx.*Minvb{i}.*Minvb{j}./(q.^2) +
           + dMinvbdx{i}.*Minvb{j}./q + Minvb{i}.*dMinvbdx{j}./q;
           obj.DMinvy{i,j} = obj.DMinvy{i,j} +
           - dqdy.*Minvb{i}.*Minvb{j}./(q.^2) +
           + dMinvbdy{i}.*Minvb{j}./q + Minvb{i}.*dMinvbdy{j}./q;
           end
         elseif i<=np && j==np+1
            obj.Minv{i,j} = -Minvb{i}./q;
            if nargin>4
               obj.DMinvx{i,j} = dqdx.*Minvb{i}./q.^2 +
```

```
                -dMinvbdx{i}./q;
                obj.DMinvy{i,j} = dqdy.*Minvb{i}./q.^2 +
                -dMinvbdy{i}./q;
            end
        elseif i==np+1 && j==np+1
            obj.Minv{i,j} = 1./q;
            if nargin>4
              obj.DMinvx{i,j} = -dqdx./q.^2;
              obj.DMinvy{i,j} = -dqdy./q.^2;
            end
         end
      end
   end
end
```

5.3 The class Mesh

The class `Mesh` contains the background integration mesh, with the Gaussian points and the weights for the domain Ω and the boundary $\partial\Omega$.

5.3.1 The properties

The main properties defined on the class `Mesh` are

- `fv` a structure where `fv.vertices` is the list of the vertices of the background mesh and `fv.faces` is the connectivity matrix that defines the elements
- `ne` is the number of *edges* that composes the boundary $\partial\Omega$
- `bfv` is an array of `ne` cells, where each cell is structure composed of vertices and faces
- `GP` is the list of coordinates of the Gaussian points in Ω
- `W` is the list of the Gaussian weights

- bGP is an array of ne cells, where each cell contains a structure composed of Gaussian points and weights
- ng is the number of the Gaussian points in Ω
- rho is the array of the dilatation parameters, calculated as the
- RKPM is the RKPM object for the Gaussian points in Ω

5.3.2 The methods

The methods defined on the class Mesh are:

1. Mesh: the *constructor* of the class. The inputs to the constructor are the name of the file containing the input data, the list of points, the vertices and the edges of the background mesh. It also calls the function RhoHex that calculates the dilatation parameter as $\rho_s\times$ the maximum length of the edges of the elements that has the $I-th$ node as vertex. ρ_s is a factor which is usually 2 for quadratic basis or 2.4 for cubic basis. Moreover, the function GaussQuad is called, that creates the Gaussian points and the weight on the triangular mesh using the mapping

$$x = x_1\phi_1(\xi,\eta) + x_2\phi_2(\xi,\eta) + x_3\phi_3(\xi,\eta), \tag{5.1}$$

$$y = y_1\phi_1(\xi,\eta) + y_2\phi_2(\xi,\eta) + y_3\phi_3(\xi,\eta), \tag{5.2}$$

where (x_1, y_1), (x_2, y_2), (x_3, y_3) are the vertices of the element, ξ and η are the local coordinates of the unit simplex and ϕ_1, ϕ_2 and ϕ_3 are the finite element shape functions deriving from Lagrangian interpolation.

$$\phi_1(\xi,\eta) = \xi, \tag{5.3}$$

$$\phi_2(\xi,\eta) = \eta, \tag{5.4}$$

$$\phi_3(\xi,\eta) = 1 - \xi - \eta. \tag{5.5}$$

The determinant of the Jacobian of equations (5.1) and(5.2) is

$$\left|\frac{\partial(x,y)}{\partial(\xi,\eta)}\right| = (x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3) = 2A, \tag{5.6}$$

where A is the area of the triangular element

2. `MeshB`: behaves as `Mesh` but on a specified edge of the boundary, using the mapping

$$x = x_1\phi_1(\xi) + x_2\phi_2(\xi), \tag{5.7}$$

$$y = y_1\phi_1(\xi) + y_2\phi_2(\xi), \tag{5.8}$$

where (x_1, y_1), (x_2, y_2) are the vertices of the element, ξ is the local coordinates of the interval $[-1, 1]$ and ϕ_1, ϕ_2 are the finite element shape functions deriving from Lagrangian interpolation

$$\phi_1(\xi) = \frac{1-\xi}{2},\ \phi_2(\xi) = \frac{1+\xi}{2}. \tag{5.9}$$

The length of the tangent vector in equations (5.7) and (5.8) is

$$\left|\frac{\partial(x,y)}{\partial\xi}\right| = \sqrt{\left(\frac{x_2-x_1}{2}\right)^2 + \left(\frac{y_2-y_1}{2}\right)^2}. \tag{5.10}$$

3. `PlotBoundary` and `PlotMesh` are methods that plot the mesh and the boundary as *patches*. A `patch` is a graph object predefined in `Matlab`, that uses the same vertices-faces structure defined in `fv` and `bfv`;

4. `PlotGP` and `PlotOnGP(obj,fun)` plot the Gaussian Points and a function `fun` on the Gaussian points as *scatter* plots.

5. `WDPHIx`, `WDPHIy`, `WPHI`, `IPHI`, `IPHIf`, `WbPHI`, `wbPHI` are methods that calculate the integrals using the `asb_handle` as in command (4.93).

5.4 The class `Load`

The properties defined in class `Load` are the natural boundary conditions (loads and displacements) applied to $\partial\Omega$, while the methods calculate the force vectors and plot the natural boundary conditions.

5.4.1 The properties

The properties for the class `Load` are:

- `DistributedTractions`: a structure whose fields are the function handles `tx` and `ty`, that contain the functional expressions for the applied distributed tractions;
- `DistributedDisplacements`: a structure whose fields are the function handles `dx` and `dy`, that contain the functional expressions for the applied distributed displacements;
- `PointForces`: a structure whose fields are: `P` coordinates of the points where concentrated forces are applied, `tbar` an array containing the components of the forces;
- `PointDisplacements`: a structure whose fields are: `P` coordinates of the points where concentrated displacements are applied, `tbar` an array containing the components of the displacements;
- `NBC`: an array $2 \times$ `ne` of logical, where 1 indicates an active component of the distributed tractions on the $j-th$ edge $j = 1, \ldots,$ `ne` and 0 otherwise;
- `DBC`: same as `NBC` for the applied distributed displacements.

5.4.2 The methods

The main methods defined on `Load` are:

1. `ApplyDistrTrac(obj,obj2,ne,tx,ty)`: calculates the distributed tractions `tx` and `ty` on the `ne`-th edge of $\partial\Omega$, where `obj2` is a `Mesh` object. The output is the force vector $\int_\Gamma \phi^T \mathbf{t} d\Gamma$ in equation (3.104). In order to calculate the outputs, the `MeshB` method is called to create the Gaussian points and the weights on the edge, then a `RKPM` object is constructed on these Gaussian points and the method `IPHI` of the `RKPM` object is invoked for the integral.

2. `ApplyDistrDispl(obj,obj2,ne,dx,dy)`: same as `ApplyDistrTrac` but for the distributed displacement. The output is the generalized force vector in $\int_{\Gamma_u} \phi^T \bar{\mathbf{u}} d\Gamma_u$ in equation (3.104) and the matrix $\int_{\Gamma_u} \phi_i \phi_j d\Gamma_u$ in equation (3.103). The method `WbPHI` is used to calculate the matrix.

3. `ApplyPointForces` same as `ApplyDistrTrac` but for concentrated forces;

4. `ApplyPointDispl` same as `ApplyPointForces` but for concentrated displacements;

5. `DeleteDistrTrac`, `DeleteDistrDispl`, `DeletePointForce` and `DeletePointDispl` are *destructors*, i.e. methods that allow to remove the applied forces and displacements;

6. `PlotLoads` and `PlotDisplacements` are graphic methods that plot the applied loads and displacements on the boundary $\partial\Omega$.

5.5 The class `Constraint`

The properties defined in class `Load` are the essential boundary conditions applied to $\partial\Omega$, while the methods calculate the matrix and plot the constraints.

5.5.1 The properties

The properties defined in the class `Constraint` are:

- `Points` structure similar to the one for the `Load` class (subsection 5.4.1). It contains the informations related to essential boundary conditions applied in points.

- `EBC` array similar to `NBC` in `Load` class (subsection 5.4.1): 1 indicates an active constraint, otherwise the value is set to 0.

5.5.2 The methods

The methods defined in the class `Constraint` are similar to the ones defined for the class `Load`:

1. `ApplyConstraints(obj,obj2,ne,c)`: applies essential boundary conditions on the `ne`-*th* edge. The output is the matrix $\int_{\Gamma_u} \phi_i \phi_j d\Gamma_u$ in equation (3.103). The method `WbPHI` is used to calculate the matrix;

2. `ApplyPointConstraints`: same as `ApplyConstraints` but on points rather than segments.

5.6 Other classes

The remaining classes are auxiliary to the main one `LinearElastic2D`, described later. The class `ElasticProperties` reads the elastic properties from the input file and calculates the *stiffness tensor* for two-dimensional linear orthotropic materials as in section 4.5. The only method implemented is the computation of equation (4.98).

The class `PostProcessing` extracts the solution of an elastic problem and writes the files containing the displacements, the stress σ and strain ϵ tensors in the Gaussian points. These data can be subsequently used for a deformation plot or for a fringe plot of the stress or the strain.

If a `Cohesive` object 6.4.9 in given as additional input, a `PostProcessing` class calculates the enriched approximation as in equations (6.73), (6.76), (6.77) and (6.78)

5.7 The class `LinearElastic2D`

In this section it is finally described the class that comprehends the above described classes. The instance created from this class is the object `Plate`.

The object behaves like an actual plate, where forces, displacements and constraints can be applied and the resulting deformed configuration (along with the levels of stress or strain) can be displayed.

5.7.1 The properties

The main properties defined on the class are:

- `Elastic`: the `ElasticProperties` object described in subsection 5.6;
- `Omega`: a `Mesh` object;
- `Loads`: a `Load` object;
- `Constraints`: a `Constraint` object;
- `Results`: a `PostProcessing` object;
- `K`: the stiffness matrix in equation (3.103);
- `Vinc`: the matrix output of the method `ApplyConstraints`;
- `Vinc_p` : the matrix output of the method `ApplyPointConstraints`;
- `Vincd_p`: the matrix output of the method `ApplyPointDispl`;
- `Vincd` : the matrix output of the method `ApplyDistrDispl`;
- `F`: the vector output of the method `ApplyDistrTrac`;
- `Fd`: the vector output of the method `ApplyDistrDispl`;
- `F_p`: the vector output of the method `ApplyPointForces`;
- `Fd_p`: the vector output of the method `ApplyPointDispl`.

5.7.2 The methods

The main methods of the class are

- `LinearElastic2D`: the *constructor*, which creates the objects `Elastic`, `Omega`, `Loads` and `Constraints` by calling the correspondent classes;

- `ApplyDistrDispl`, `ApplyDistrTrac`, `ApplyPointForces`, `ApplyPointDispl`, `ApplyConstraints` and `ApplyPointConstraints` are overloaded methods from the classes `Load` and `Constraint` for the application of forces and constraints;

- `DeleteDistrTrac`, `DeleteDistrDispl`, `DeletePointForces`, `DeletePointDisplacements`, `DeleteConstraints` and `DeletePointConstraints` are overloaded methods from the classes `Load` and `Constraint` for the deletion of forces and constraints;

- `PlotLoads`, `PlotApplDisplacements` `PlotConstraints` are overloaded methods from the classes `Load` and `Constraint` for the plot of forces and constraints;

- `PlotMesh`, `PlotOnGP` are overloaded methods from the class `Mesh` for the plot of the mesh and the Gaussian points;

- `Assembly`: method that creates the stiffness matrix $\mathbf{K}$ as described in section 4.5; It creates an object of the class `RKPM` for the shape functions in the Gaussian points `GP` contained in the object `Omega` of the class `Mesh`;

- `PostProcessing`: overloaded constructor of the object `Results`;

- `Solve` is the method that solves the linear elastic problem once the stiffness matrix and the force vectors are calculated:

```
R = (obj.K+obj.Options.Penalty*obj.Vinc+
+obj.Options.Penalty*obj.Vinc_p+
+obj.Options.Penalty*obj.Vincd_p +
+obj.Options.Penalty*obj.Vincd )\(obj.F +
+ obj.F_p + obj.Options.Penalty*obj.Fd +
+ obj.Options.Penalty*obj.Fd_p);
```

  . The solution of the linear system of equations is performed by the *built-in* operator \, that automatically chooses the most suitable algorithm according to the structure of the matrix `K`. When the solution is completed, the object `Results` is created

```
obj.Results = PostProcessing(R(1:obj.Omega.ns,:),...
...R(obj.Omega.ns+1:2*obj.Omega.ns,:));
```

.

5.8 The class LinearElastic3D

The three-dimensional class is very similar to the two-dimensional class. The difference lies in the class `Mesh` that makes use of predefined classed in the newest releases of `Matlab` as `TriRep` and `DelaunayTri`.

`TriRep` provides topological and geometric queries for triangulations in 3-D space. For example, for tetrahedral meshes numerous queries can be made, like triangles attached to a vertex, triangles that share an edge, neighbour information, circumcenters, or other features. An object of the class `TriRep` can be created directly using existing triangulation data. If only points are available and a triangulation is required, the class `DelaunayTri` can be used instead. `DelaunayTri` creates a Delaunay triangulation when coordinates of points are inserted as input. Also, it contains a `listener` that automatically modifies the triangulation whenever new points are inserted. The figures 5.1,5.2 and 5.3 illustrates the capabilities of the 3D code. The construction of the Gaussian points is different, because a different mapping is necessary.

For the tetrahedral elements:

$$x = x_1\phi_1(\xi,\eta,\zeta) + x_2\phi_2(\xi,\eta,\zeta) + x_3\phi_3(\xi,\eta,\zeta) + x_4\phi_4(\xi,\eta,\zeta), \tag{5.11}$$

$$y = y_1\phi_1(\xi,\eta,\zeta) + y_2\phi_2(\xi,\eta,\zeta) + y_3\phi_3(\xi,\eta,\zeta) + y_4\phi_4(\xi,\eta,\zeta), \tag{5.12}$$

$$z = z_1\phi_1(\xi,\eta,\zeta) + z_2\phi_2(\xi,\eta,\zeta) + z_3\phi_3(\xi,\eta,\zeta) + z_4\phi_4(\xi,\eta,\zeta), \tag{5.13}$$

where (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4) are the vertices of the element, ξ, η and ζ are the local coordinates of the unit simplex and ϕ_1, ϕ_2, ϕ_3 and

ϕ_4 are the finite element shape functions deriving from Lagrangian interpolation.

$$\phi_1(\xi, \eta, \zeta) = \xi, \tag{5.14}$$
$$\phi_2(\xi, \eta, \zeta) = \eta, \tag{5.15}$$
$$\phi_3(\xi, \eta, \zeta) = \zeta, \tag{5.16}$$
$$\phi_4(\xi, \eta, \zeta) = 1 - \xi - \eta - \zeta. \tag{5.17}$$
$$\tag{5.18}$$

The determinant of the Jacobian of equations (5.11), (5.12) and (5.13)is

$$\left| \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \right| = 6V; \tag{5.19}$$

where V is the volume of the tetrahedral element.

For the triangular elements of the surfaces of the boundaries, the following mapping is used

$$x = x_1\phi_1(\xi, \eta) + x_2\phi_2(\xi, \eta) + x_3\phi_3(\xi, \eta), \tag{5.20}$$
$$y = y_1\phi_1(\xi, \eta) + y_2\phi_2(\xi, \eta) + y_3\phi_3(\xi, \eta), \tag{5.21}$$
$$z = z_1\phi_1(\xi, \eta) + z_2\phi_2(\xi, \eta) + z_3\phi_3(\xi, \eta), \tag{5.22}$$
$$\tag{5.23}$$

where (x_1, y_1), (x_2, y_2) and (x_3, y_3, z_3) are the vertices of the element, ξ and η are the local coordinates of the unit simplex and ϕ_1, ϕ_2 and ϕ_3 are the finite element shape functions deriving from Lagrangian interpolation. The Jacobian used in the mapping is

$$\left| \frac{\partial(x, y, z)}{\partial\xi} \times \frac{\partial(x, y, z)}{\partial\eta} \right| = 2A, \tag{5.24}$$

where A is the area of the triangle.

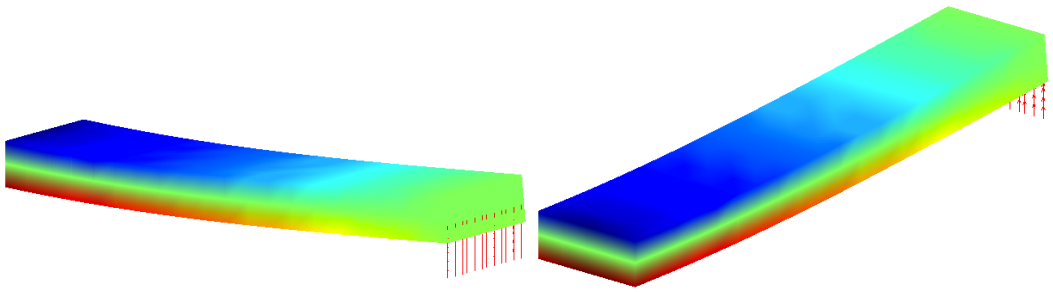
Finally, for the linear elements of the edges, the following mapping is used

$$x = x_1\phi_1(\xi) + x_2\phi_2(\xi), \tag{5.25}$$
$$y = y_1\phi_1(\xi) + y_2\phi_2(\xi), \tag{5.26}$$
$$z = z_1\phi_1(\xi) + z_2\phi_2(\xi), \tag{5.27}$$

where (x_1, y_1), (x_2, y_2) and (x_3, y_3) are the vertices of the element, ξ is the local coordinates of the interval $[-1, 1]$ and ϕ_1, ϕ_2 are the finite element shape functions deriving from Lagrangian interpolation.

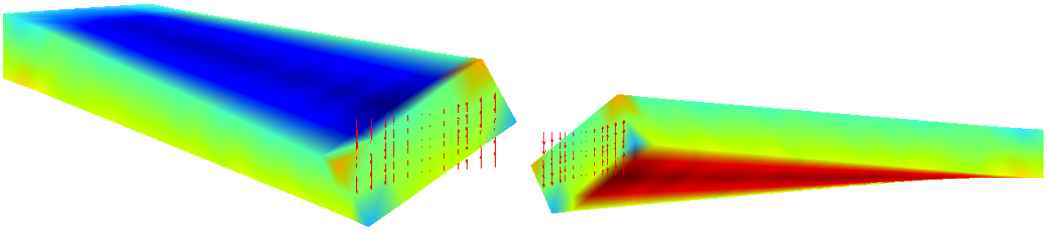
The length of the vector in equations (5.25) and (5.26) and (5.27) is

$$\left|\frac{\partial(x, y, z)}{\partial \xi}\right| = \sqrt{\left(\frac{x_2 - x_1}{2}\right)^2 + \left(\frac{y_2 - y_1}{2}\right)^2 + \left(\frac{z_2 - z_1}{2}\right)^2} \tag{5.28}$$



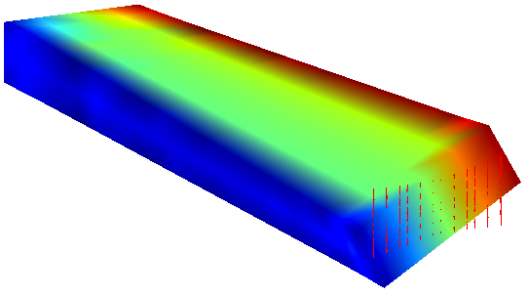
(a) Longitudinal stress (view 1) (b) Longitudinal stress(view 2)

Figure 5.1: Three-dimensional code: vertical bending



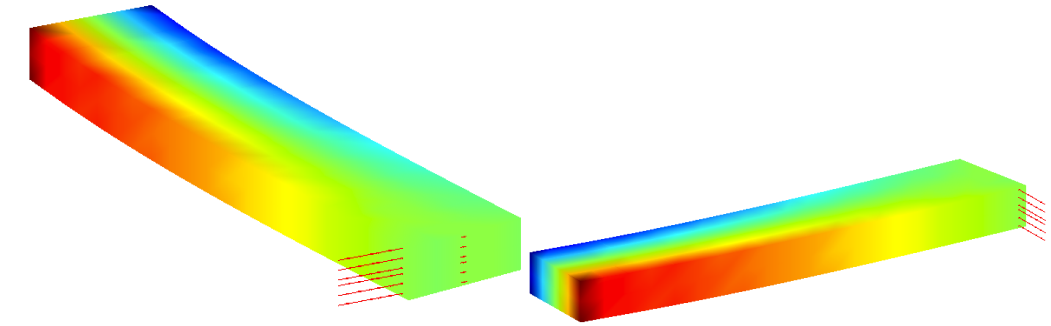
(a) In-plane shear stress

(b) In-plane shear stress



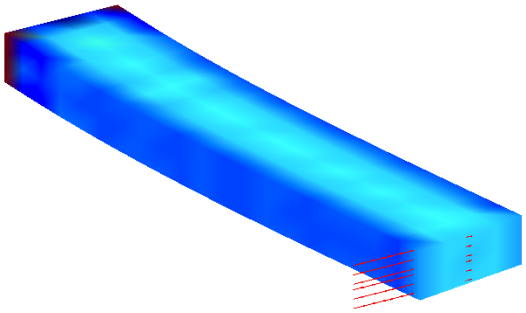
(c) Out-of-plane shear stress

Figure 5.2: Three-dimensional code: torsion

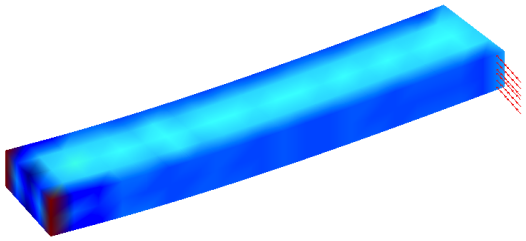


(a) Longitudinal stress (view 1)

(b) Longitudinal stress (view 2)



(c) In-plane shear stress (view 1)



(d) In-plane shear stress (view 2)

Figure 5.3: Three-dimensional code: lateral bending

Chapter 6

Application to Composite Materials: Delamination

In this chapter the meshfree techniques and the codes presented in the previous chapters will be used to model failure in composite materials. With the term *failure* is usually intended the event that occurs when the structure loses its ability to withstand an external load. This load may be static or dynamics, periodic or impulsive and the global/local response of the material might vary according to its mechanical properties. Therefore by the word *failure* a broad class of problems may be intended.

These problems may be classified by their dependency with time (static, quasi-static, creep, fatigue, dynamic loading), or by the stress-strain constitutive relationship (classic elastic fracture mechanics, plastic fracture mechanics,), or by their ultimate behaviour (brittle or ductile materials). Moreover, materials can have different behaviour if subjected to high-rate loads. For example metals are ductile at quasi-static loads, but turn into brittle if subjected to shock loads. Also, the opposite may occur. It is for example the case of concrete, which is brittle for quasi static loads, but ductile at elevated rates.

Simulation of these phenomena may require quite complex computational tools, not yet completely developed, since it can be done bridging different length scales, as in the so called *multi-scale* methods. The advent of new technologies with faster computers is fostering the application of these methodologies also to large scale simulations.

Before proceeding further, it is then opportune to define and restrict the type of failure that will be modeled with the presented method. Simulation of failure will not be focused on *multi-scale* method, but rather on a more engineering level. At a practical level, for the simulation of failure two approaches can be followed, a *damage mechanics* based and a *fracture mechanics* based.

These two approaches are not mutually exclusive, but rather complementary branches of the solid mechanics. Fracture mechanics utilizes balance laws (J-integral Rice (1968)) or state variable derived criteria (Stress Intensity Factors) to predict the failure, without modifying (introducing additional internal variables) the underlying constitutive model(s) but rather works "on top" of them. Damage mechanics [1], however, uses thermodynamics concepts. CDM introduces a new state variable (the *damage*, be it scalar, vector, matrix, tensor) to the constitutive model along with its evolution equation (and failure/initiation criteria etc.) and coupling to other state variables. Damage mechanics is more suitable for the prediction of the onset, while fracture mechanics is based on the pre-existence of a crack. The method utilized in the proceeding of the chapter is a damage mechanics based method combined to a fracture mechanics approach, able to represent geometric discontinuities. Following damage mechanics, it is crucial the choice of a correct constitutive model.

Nevertheless, in composite materials, the inhomogeneity introduces further complexity and uncertainty to the derivation of constitutive models. Damage initiation and propagation may be therefore completely different from the ones in homogeneous materials. In fact matrix and fibers usually have independent failure mechanisms and thresholds. On the top of that, bonding limits between the individual components along their interfaces represent *new* modes of failure.

Generally, two categories of failure are predominant in laminated composite materials, i.e. *intra-laminar* failures (for example matrix failure, fibers breaking, interface failure between matrix and fibers) and inter-laminar failure, i.e. *delamination*. A complete review of the numerous failure theories available for composite materials can be found in Orifici *et al.* (2008).

For fibers breaking there exist various criteria to predict the tensile failure, for example Puck & Schürmann (2002). The breaking is due to accumulation of

[1] More often referred as *Continuum Damage Mechanics* CDM

each fibre within the ply and different criteria for the compressive failure, due to microbuckling and formation of kink bands. Moreover, there are methods that do not distinguish between the two types of failure, but use a global criterion that comprises both tension and compression.

Matrix failure is due to coalescence of accumulated micro-cracks, typically originated at defects or at fibre-matrix interface. Eventually this coalescence leads to failure across a critical fracture plane. This plane can be predicted using fracture mechanics tools. Examples of criteria can be found for example in Hashin (1980) and Puck & Schürmann (2002).

The above mentioned mechanisms however will not be covered by the approaches developed in the the following sections. The main focus of this work will be *delamination*, although the previous mentioned failure mechanism could constitute the basis of future work, perhaps in a multiscale perspective.

Delamination is the debonding of the layers that may occur for example in case of low velocity impact or from a manufacturing defect such shrinkage of the matrix during curing. It is an important cause of failure in unidirectional fiber-reinforced composites. Since there is no reinforcement in the direction perpendicular to the layers, the plies are more exposed to failure in case of impact. The major inconvenient resulting from delamination is a consistent reduction in the load carrying capability of a structure.

There are mainly two types of delamination, as reported in Bolotin (1996) and Bolotin (2001): near-surface delamination and internal delaminations. Near-surface delaminations are localized on the surface of the laminate and their deformation is not much influenced by the rest of the laminate. The accurate simulation of these delaminations prescribes the analysis of the local stability.

This chapter will be focused though on internal delaminations, with an existing pre-crack. As in fracture mechanics, three modes of failure exist for internal delamination, as depicted in figure 6.1. Mode I is called *opening mode*, since separation occurs perpendicularly to the loading force, while the other two are *shear* modes. In the shear modes, the separation is parallel to the loading, but while mode II is a sliding mode (in-plane shear stress), mode III is an *out-of-plane* shear stress loading mode.

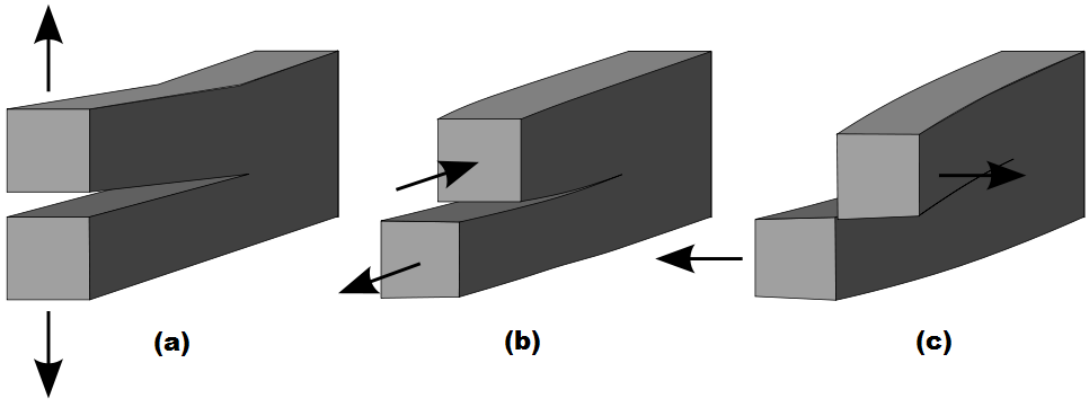


Figure 6.1: Failure modes for internal delamination: (a) mode I , (b) mode II , (c) mode III

6.1 Review of simulation of delamination

The simulation of delamination in composites is usually divided into delamination initiation and delamination propagation. The first one is usually based on the *strength of materials* theory and employs stress criteria normally based on point or average stress, while propagation is modeled with *fracture mechanics* tools based on the evaluation of the energy release rate G.

In the past years finite elements (FE) have proved to be the most widely used computer resource in the industrial and scientific community to model delamination. Particularly, *decohesion elements* are the most popular choice and they are nowadays implemented in many commercial FE packages. The main advantage of the use of decohesion elements is the capability to predict both onset and propagation of delamination. These elements could be of different types, like *continuous interface elements* (zero-thickness or finite-thickness volumetric elements) or *point decohesion elements* (non-linear springs connecting nodes). Decohesion elements use the *cohesive model* as the constitutive relationship between the relative displacements and the traction forces at the interfaces Camanho *et al.* (2001), Camanho & Dávila (2002), Camanho *et al.* (2003). Cohesive model will be described in detail in section 6.2. Finite elements methods, like decohesion elements or interface elements, have the major disadvantage that crack path is highly dependent on the mesh structure, since discontinuities must

follow inter-element boundaries, therefore capturing of a propagating discontinuity is achieved by constant re-meshing of the structure. This procedure is costly not to mention the degradation of the accuracy of the solution.

These limitations are overcome with novel methods such as meshfree or other *Partition of Unity*-based methods (PUM), where discontinuities are not restricted to element boundaries or even no elements are necessary at all. An important progress in this sense has been recently obtained with the introduction of *partition of unity*-based methods (PUM) Babuska & Melenk (1997) like *extended finite element* (XFEM) Moes *et al.* (1999), Black & Belytschko (1999), Dolbow *et al.* (2000), Moës & Belytschko (2002) and meshless methods like Element-Free Galerkin Belytschko *et al.* (1994b) or Reproducing Kernel Particle Method (RKPM) Liu *et al.* (1995). In the papers of Wells & Sluys (2001) and Remmers *et al.* (2003a), a new method called *cohesive segments* method is introduced. In Wells & Sluys (2001) the PUM idea is applied in the context of cohesive cracks. A full non-linear model is developed, with tractions acting on the cohesive surfaces. These tractions are depending on the opening displacement (or discontinuous displacement) and used to model crack propagation in three-points bending test and single edge notched beam.

In Remmers *et al.* (2003a) these segments are introduced under a FE framework, similarly to the Extended Finite Element Method (XFEM). Using a particular instance of a PUM , a continuous crack is approximated by a set of segments, each one of them split the domain in two parts. At each of these interfaces, a cohesive model is used in order to simulate the debonding of the parts. It can be shown that by doing so, the total displacement is *enriched* by a sum of *Heaviside* functions that can effectively represent the discontinuity. In this way, additional unknowns are introduced to the final algebraic system of equations. These segments can be introduced at any time of the calculation, whenever a stress-based failure criterion is satisfied, and the orientation of the onset cohesive segment is given by the principal direction of the stress. In this way, no explicit representation of the crack surface is needed allowing arbitrary crack growth, which is extremely useful when the crack propagation path is not known *a priori*. The drawback of this method is that additional unknowns are introduced at each crack segment. The unknowns are localized to the nodes of the elements cut by the

discontinuity. In Remmers *et al.* (2008) the same approach is used to efficiently simulate dynamic crack propagation.

The idea of combining cohesive segments with meshfree methods is a very recent idea Sun *et al.* (2007), although delamination modeling with Element Free Galerkin can be found in Guiamatsia *et al.* (2009), but combined with a Virtual Crack Closure Method rather than cohesive laws.

When the crack path is *known* a priori, like in the case of single mode delamination, the same approach can be used without adding additional unknowns, like in Barbieri & Meo (2008) and Barbieri & Meo (2009a).

In this case, the domain can be decomposed in two parts connected by a penalty parameter, as also suggested in Belytschko *et al.* (1996a). The variational problem is formulated for each sub-domain and then a coupling term between their displacements is introduced. The coupling term is nothing else than a penalty factor defined on the whole contact segment. Penalty terms are well-known in the meshfree community since, conversely to FE, the shape functions do not possess the Kronecker condition Fries & Matthies (2004). This condition allows in FE to *directly* impose essential boundary conditions on the nodes located on the constrained boundaries. For simple constraints, though, a constant penalty factor is sufficient throughout the whole boundary. The same approach can be used to *connect* or *disconnect* two or more objects.

At this aim, a penalty-based finite element technology was introduced in Pantano & Averill (2002) to assemble together independently modeled finite element sub-domains. The same technology was subsequently used in Pantano & Averill (2004) to model mixed-mode delamination growth, using the concept that a penalty factor that was previously used to bond together different parts can be used as well for their separation. The simulation is almost mesh independent, but additional unknowns are required for the interface, even for single delamination. It will be shown in the next sections that using meshfree shape functions this limitation can be removed.

The coupling term that derives from the application of the cohesive model at the separation leads to a final nonlinear set of equations

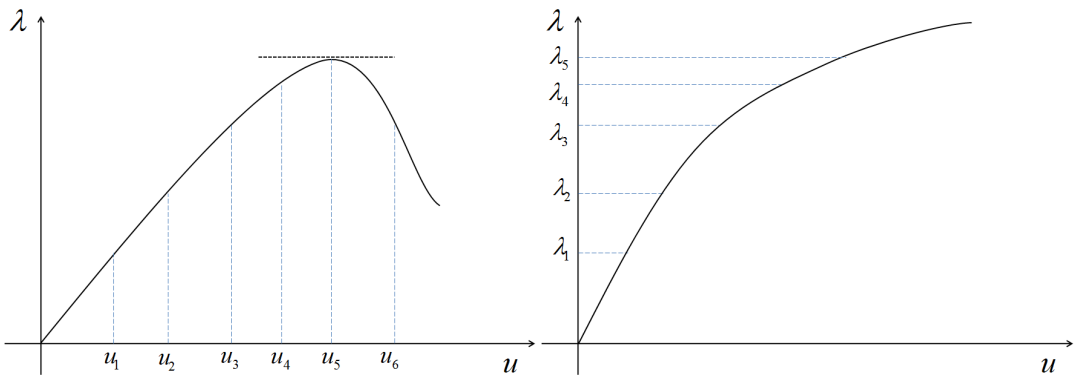
$$\mathbf{\Psi}(\mathbf{a}, \lambda) = \mathbf{R}(\mathbf{a}) - \lambda \mathbf{P} = 0, \tag{6.1}$$

where $\mathbf{a}$ is a $2n_s \times 1$ vector which is usually the displacement vector, $\mathbf{P}$ is the unitary external force vector, λ is a scalar that quantifies the magnitude of the applied force and $\mathbf{R}(\mathbf{a})$ is the *internal force* vector. Problem (6.1) is called *parametric continuation*.

In computational solid mechanics, equation (6.1) often refers to the search of an equilibrium path $(\mathbf{a}, \lambda)$ and two approaches can be followed, as reported in the classic book Crisfield (1991):

1. *displacement control*, where a displacement is applied and λ derives from a posteriori computation of a reaction force (figure 6.2a)

2. *force control* where a force is applied and λ is treated as an additional unknown (figure 6.2b)

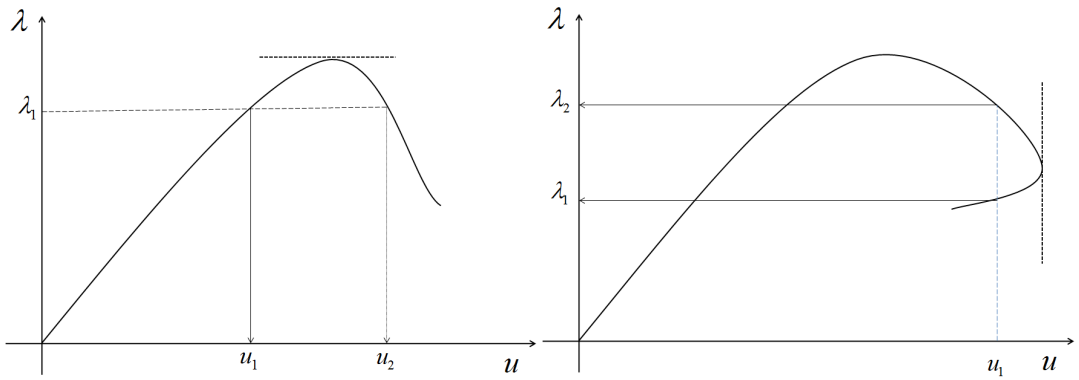


(a) Displacement Control (b) Force Control

Figure 6.2: Examples of equilibrium paths for non-linear equations

Displacement control may be suitable when the equilibrium curve presents an unstable point, where the tangent at the curve is horizontal to the displacement axis (figure 6.3a). In fact, in this case for the same load level λ there exist two different equilibrium states u_1 and u_2. Nevertheless, displacement control may fail as well for the case illustrated in figure 6.3b, where the same equilibrium state u_1 is satisfied simultaneously by two different load levels λ_1 and λ_2.

Nonetheless, the equilibrium path could be more complicated and for example in case in figure 6.3b both displacement and force control fail. A more general



(a) Failure of the force control method

(b) Failure of the displacement and force control method

Figure 6.3: Failures in the load control and displacement control

approach to trace the equilibrium path in figure 6.3b is therefore the force control with an *arc-length* solver. To provide more robustness, the *arc-length* can be also combined with a line-search approach. Following Crisfield (1991), a detailed procedure is reported in appendix B. A `Matlab` in-house code has been written to serve this scope.

The outline of the chapter is the following: firstly, in section 6.2, the cohesive zone model will be reviewed, then in section 6.3 the penalty approach to model delamination will be presented and numerical cases will be shown. Finally, in section 6.4 a more general approach will be presented, known as the *cohesive segments method* that exploits the PUM principle along with numerical cases.

6.2 Cohesive Zone Model

Cohesive models are based on the Dudgale - Barenblatt Barenblatt (1962) cohesive zone approach, subsequently extended by Hillerborg Hillerborg *et al.* (1976) and Needleman Needleman (1987).

The main purpose of cohesive zone models is to provide a more physical explanation of the failure process, differently from classical linear fracture mechanics, which lacks of a physics-based description Klein *et al.* (2001) and it is based on the concentration of stress at notches. Conversely to classical fracture mechanics,

the cohesive zone modeling approach does not involve crack tip stress singularities and failure is regulated by relative displacements and stresses.

In the cohesive zone model instead, it is postulated the existence of a narrow band of vanishing thickness ahead of a crack tip which represents the fracture process zone. The bonding of the surfaces of the zone is obtained by cohesive traction that derives from a potential. Specifically, in the work of Needleman (1987) [1], it is introduced a potential $\phi(u_n, u_t, u_b)$ function of the normal component u_n, the tangential component u_t and the binormal component u_b of the displacement jump $\mathbf{\Delta}$ at the interface (figure 6.4).

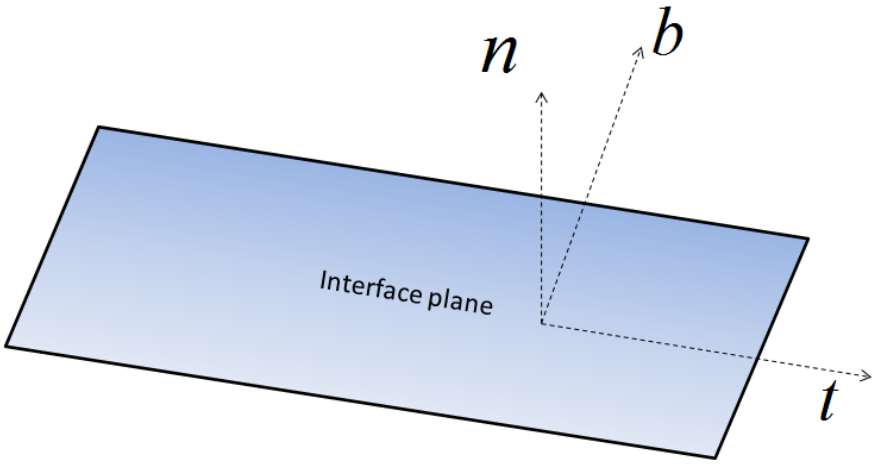


Figure 6.4: Reference frame for the interface

$$\begin{aligned}\phi(u_n, u_t, u_b) &= -\int_0^u T_n \mathrm{d}u_n + T_t \mathrm{d}_t + T_b \mathrm{d}_b = \\ &= \frac{27}{4}\sigma_{max}\left\{\left(\frac{u_n}{\delta}\right)^2\left[1 - \frac{4}{3}\left(\frac{u_n}{\delta}\right) + \frac{1}{2}\left(\frac{u_n}{\delta}\right)^2\right] + \right. \\ &\left. + \frac{1}{2}\alpha\left(\frac{u_t}{\delta}\right)^2\left[1 - 2\left(\frac{u_n}{\delta}\right) + \left(\frac{u_n}{\delta}\right)^2\right] + \frac{1}{2}\alpha\left(\frac{u_b}{\delta}\right)^2\left[1 - 2\left(\frac{u_n}{\delta}\right) + \left(\frac{u_n}{\delta}\right)^2\right]\right\} \quad u_n \leq \delta,\end{aligned} \tag{6.2}$$

where T_n, T_t and T_b are the tractions respectively on direction n, t and b and δ is a characteristic length, σ_{max} is the maximum traction carried at the interface

[1] the same notation will be used

under purely normal separation ($u_t = u_b = 0$), α the ratio between tangential and normal stiffness.

When u_n exceeds δ, then the potential ϕ equals the work of separation ϕ_{sep}. These traction can be obtained by differentiating (6.2)

$$T_n = \frac{\partial \phi}{\partial u_n} = -\frac{27}{4}\sigma_{max}\left\{\left(\frac{u_n}{\delta}\right)\left[1 - 2\left(\frac{u_n}{\delta}\right) + \left(\frac{u_n}{\delta}\right)^2\right] + \right.$$
$$\left. + \frac{1}{2}\alpha\left(\frac{u_t}{\delta}\right)^2\left[\left(\frac{u_t}{\delta}\right) - 1\right] + \frac{1}{2}\alpha\left(\frac{u_b}{\delta}\right)^2\left[\left(\frac{u_b}{\delta}\right) - 1\right]\right\}, \quad (6.3)$$

$$T_t = \frac{\partial \phi}{\partial u_t} = -\frac{27}{4}\sigma_{max}\alpha\left(\frac{u_t}{\delta}\right)\left[1 - 2\left(\frac{u_n}{\delta}\right) + \left(\frac{u_n}{\delta}\right)^2\right], \quad (6.4)$$

$$T_b = \frac{\partial \phi}{\partial u_b} = -\frac{27}{4}\sigma_{max}\alpha\left(\frac{u_b}{\delta}\right)\left[1 - 2\left(\frac{u_n}{\delta}\right) + \left(\frac{u_n}{\delta}\right)^2\right]. \quad (6.5)$$

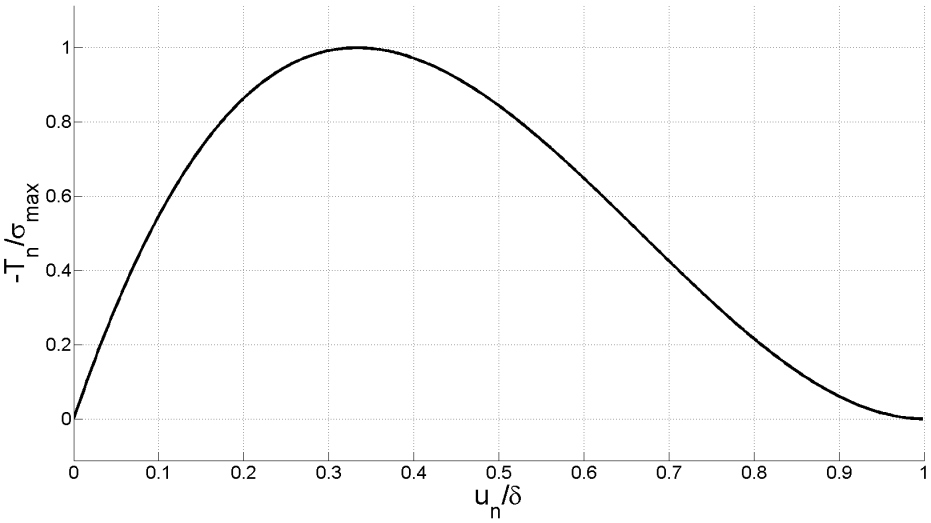


Figure 6.5: Normal traction T_n - relative displacement u_n curve for $\alpha = 0$ (equation (6.3))

The work of separation (or critical strain energy release rate G_c) is

$$\phi_{sep} = \frac{9}{16}\sigma_{max}\delta \quad (6.6)$$

This model, though, describes only debonding by normal separation. In Tvergaard (1990) this model was generalized by introducing a damage variable D. In addition, a cleavage unloading procedure can be added, in which the stiffness is reduced when damage occurs but separation totally vanishes when the traction is reduced to zero.

$$D = \sqrt[m]{\left(\frac{u_t}{\delta_{max}}\right)^m + \left(\frac{u_n}{\delta_{max}}\right)^m} \qquad D \in [0,1], \tag{6.7}$$

when $m = 2$, then the tractions are

$$T_n = \frac{27}{4}\sigma_{max}\frac{u_n}{\delta}(1-D)^2, \tag{6.8}$$

$$T_t = \frac{27}{4}\sigma_{max}\frac{u_t}{\delta}(1-D)^2. \tag{6.9}$$

Many different constitutive laws exist, for example the *exponential* or the *bilinear* softening model Zou *et al.* (2003). These laws are typically regulated by both strength of material parameters (like the tensile strength) and fracture mechanics based parameters like the critical fracture energies for the considered fracture mode. Crack growth occurs when a critical value is reached at which cohesive traction disappears.

The cohesive model implemented in the following sections is a bilinear traction - displacement relationship $\tau(\boldsymbol{\Delta})$, where τ is the traction at the interface and $\boldsymbol{\Delta}(\mathbf{x})$ is the displacement jumps at the interface. The *bilinear* curve is divided into three main parts (fig. 6.6) and the constitutive equations are the following:

- $\Delta \leq \Delta_0$: *elastic part*: traction across the interface increases until it reaches a maximum, and the stress is linked to the relative displacement via an arbitrary interface stiffness K_0 :

$$\tau(\Delta) = K_0\Delta; \tag{6.10}$$

- $\Delta_0 < \Delta \leq v_F$: *softening part*: the traction across the interface decreases until it becomes equal to zero: the two layers begin to separate. The damage accumulated at the interface is represented by a variable D, which is equal

to zero when there is no damage and reaches 1 when the material is fully damaged:

$$\tau(\Delta) = (1 - D(\Delta))K_0\Delta; \tag{6.11}$$

- $\Delta > \Delta_f$: *decohesion part*: decohesion of the two layers is complete: there is no more bond between the two layers, the traction across the interface is null.

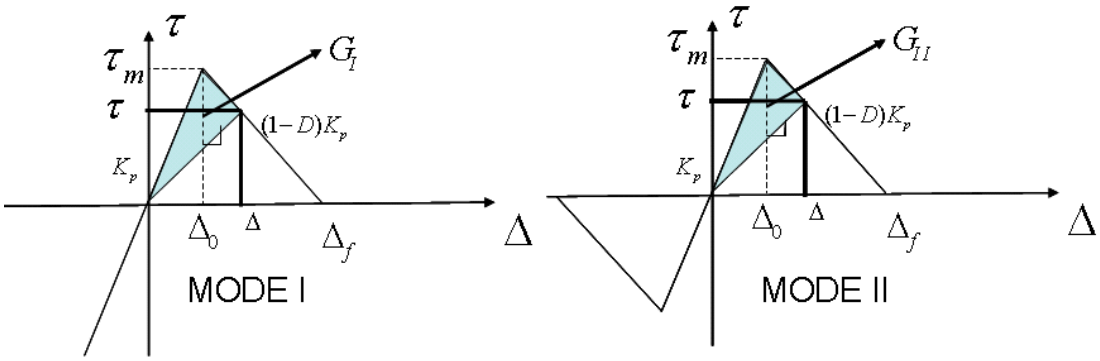


Figure 6.6: Bilinear softening model

The shaded area in figure 6.6 is the energy dissipated per unit area G for a particular mode. Moreover, for mode II the diagram is symmetrical with respect to the zero, since the sliding motion could occur in both senses (figure 6.1), while to prevent inter-penetration of the crack surfaces, diagram for mode I is linear for negative relative displacements Δ and no damage is prescribed. In this way, the stiffness parameter is restored in order to keep the interface *closed*.

When $\Delta = \Delta_f$ the area of the whole triangle is the critical energy dissipated per unit area G_c. It can be shown Rice (1968) that in this case cohesive zone approaches can be related to Griffith's theory of fracture. Moreover Alfano & Crisfield (2001) showed that when $\Delta_0 = \Delta_f$ (which means abrupt load fall to zero) a perfectly brittle fracture can be simulated.

Two independent parameters are necessary to define a bilinear softening model, i.e. the inter-facial maximum strength τ_m and the critical energy dissipated per

unit area G_c, since the following relations hold

$$G_c = \frac{\tau_m \Delta_f}{2} \tag{6.12}$$

$$\tau_m = K_0 \Delta_0 \tag{6.13}$$

where K_0 is an arbitrary initial *penalty stiffness* (dimensionally N/m^3) which is usually set as a large number. Then, once derived Δ_0 and Δ_f, the variable damage D can be calculated as

$$D(\Delta) = \begin{cases} 0 & \Delta \leq \Delta_0 \\ \frac{\Delta_f(\Delta - \Delta_0)}{\Delta(\Delta_f - \Delta_0)} & \text{if } \Delta_0 < \Delta \leq \Delta_F \\ 1 & \Delta > \Delta_F \end{cases} \tag{6.14}$$

6.3 The Cohesive Penalty approach

In the *penalty* approach, Barbieri & Meo (2009a), for single loading modes I and II, the two layers are modelled as two independent beams, Ω_1 and Ω_2. The key parameter of a bilinear cohesive model, i.e. the arbitrary penalty stiffness, is defined over the contact surface Γ_c (figure 6.7) and weakened according to a cohesive law.

6.3.1 The equations of equilibrium

Assuming that there are no body or inertia forces, the strong form of the equilibrium equations along with the boundary conditions can be written as

$$\nabla \cdot \sigma = 0 \qquad \mathbf{x} \in \Omega, \tag{6.15}$$

$$\mathbf{n}_t \cdot \sigma = \bar{\mathbf{t}} \qquad \mathbf{x} \in \Gamma_t, \tag{6.16}$$

$$\mathbf{u} = \bar{\mathbf{u}} \qquad \mathbf{x} \in \Gamma_u, \tag{6.17}$$

$$\mathbf{n}_c \cdot \sigma = \tau(\boldsymbol{\Delta}) \qquad \mathbf{x} \in \Gamma_c, \tag{6.18}$$

where Ω is the entire domain, σ is the Cauchy stress tensor, $\mathbf{n}_t$ is the normal unity vector of the boundary Γ_t where the traction $\bar{\mathbf{t}}$ is prescribed, $\mathbf{n}_c$ is the normal unity vector of the boundary Γ_c where the traction $\tau(\boldsymbol{\Delta})$ is prescribed and Γ_u is the boundary where the displacement $\bar{\mathbf{u}}$ is imposed.

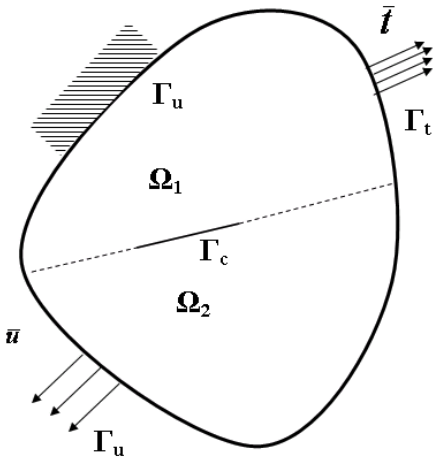


Figure 6.7: Description of the problem

The traction τ depends on the displacement jump $\boldsymbol{\Delta}$ on the cohesive segment Γ_c, which divides Ω in two sub-domains Ω_1 and Ω_2 as in figure 6.7. The displacement jump $\boldsymbol{\Delta}(\mathbf{x})$ is defined as follows

$$\boldsymbol{\Delta}(\mathbf{x}) = \mathbf{u}_1(\mathbf{x}) - \mathbf{u}_2(\mathbf{x}) \qquad \mathbf{x} \in \Gamma_c, \tag{6.19}$$

where $\mathbf{u}_1(\mathbf{x})$ is the displacement of sub-domain Ω_1 and $\mathbf{u}_2(\mathbf{x})$ is the displacement of sub-domain Ω_2.

Using displacement $\mathbf{u}$ as test function for equations (6.15), (6.16) and displacement jump $\boldsymbol{\Delta}$ as test function for equation (6.18), the variational principle can be written as

$$\int_{\Omega_1} \delta\epsilon^T \sigma d\Omega_1 + \int_{\Omega_2} \delta\epsilon^T \sigma d\Omega_2 - \int_{\Gamma_{t1}} \delta\mathbf{u}^T \bar{\mathbf{t}} d\Gamma_{t1} - \int_{\Gamma_{t2}} \delta\mathbf{u}^T \bar{\mathbf{t}} d\Gamma_{t2} + \frac{\alpha}{2}\delta \int_{\Gamma_{u1}} (\mathbf{u} - \bar{\mathbf{u}})^2 d\Gamma_{u1} + \frac{\alpha}{2}\delta \int_{\Gamma_{u2}} (\mathbf{u} - \bar{\mathbf{u}})^2 d\Gamma_{u2} + \int_{\Gamma_c} \delta\boldsymbol{\Delta}^T \tau d\Gamma_c = 0, \tag{6.20}$$

where the penalty method is used to enforce essential boundary conditions (6.17) and α is called *penalty parameter*, which is usually an arbitrary very large number. After having performed several numerical experiments, a penalty factor of $1e15$ seemed to efficiently enforce essential boundary conditions without causing instability.

Varying this number it is possible to loosen or tighten a certain constraint. This will be useful in the next sections with the application of the cohesive model

at the crack interface Γ_c. In equation (6.20) Γ_{t1} and Γ_{u1} refer to the part of boundaries Γ_u and Γ_t that belong to sub-domain Ω_1, whereas Γ_{t2} and Γ_{u2} the ones that belong to Ω_2.

6.3.2 Discretization of equations of equilibrium

In general, for a two-dimensional case, (but similar argument can be conducted in the three-dimensional case), naming t the tangential direction the cohesive segment and n the normal direction, the stress - relative displacement relationship can be formulated as

$$\tau = \begin{bmatrix} \tau_t \\ \tau_n \end{bmatrix} = \begin{bmatrix} K_t(\Delta_t, \Delta_n) & 0 \\ 0 & K_n(\Delta_t, \Delta_n) \end{bmatrix} \begin{bmatrix} \Delta_t \\ \Delta_n \end{bmatrix} = \\ = \begin{bmatrix} (1 - D_t(\Delta_t, \Delta_n))K_{0t} & 0 \\ 0 & (1 - D_n(\Delta_t, \Delta_n))K_{0n} \end{bmatrix} \begin{bmatrix} \Delta_t \\ \Delta_n \end{bmatrix}. \quad (6.21)$$

If a given loading mode has reached 1, the damage variable corresponding to the other loading mode is set as well to 1 to avoid that the material would still be able to carry tractions.

In the following examples t corresponds to axis x and n to axis y. For arbitrary orientations, a transformation matrix would be necessary to express displacements from the global reference frame to the local one (figure 6.8).

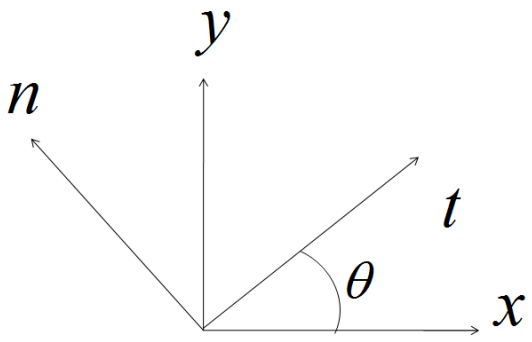


Figure 6.8: Reference transformation from global coordinates to local coordinates

Displacement jump in equation (6.19) can be then expressed as

$$\boldsymbol{\Delta} = \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} = \begin{bmatrix} \phi_{1c}^T \mathbf{U}_1 - \phi_{2c}^T \mathbf{U}_2 \\ \phi_{1c}^T \mathbf{V}_1 - \phi_{2c}^T \mathbf{V}_2 \end{bmatrix} = \begin{bmatrix} \phi_{1c}^T & 0 & -\phi_{2c}^T & 0 \\ 0 & \phi_{1c}^T & 0 & -\phi_{2c}^T \end{bmatrix} \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{V}_1 \\ \mathbf{U}_2 \\ \mathbf{V}_2 \end{bmatrix} = \tilde{\phi}^T \mathbf{R}, \tag{6.22}$$

where

$$\phi_{1c} = \phi_1(\mathbf{x}) \qquad \mathbf{x} \in \Gamma_c, \tag{6.23}$$

where $\phi_1(\mathbf{x})$ are the shape functions of sub-domain Ω_1, whereas

$$\phi_{2c} = \phi_2(\mathbf{x}) \qquad \mathbf{x} \in \Gamma_c \tag{6.24}$$

are the shape functions belonging to sub-domain Ω_2,

$$\tilde{\phi}^T = \begin{bmatrix} \phi_{1c}^T & 0 & -\phi_{2c}^T & 0 \\ 0 & \phi_{1c}^T & 0 & -\phi_{2c}^T \end{bmatrix} \tag{6.25}$$

and $\mathbf{R}_1^T = \begin{bmatrix} \mathbf{U}_1{}^T & \mathbf{V}_1{}^T \end{bmatrix}$ is the displacement vector for nodes located in Ω_1 whereas $\mathbf{R}_2^T = \begin{bmatrix} \mathbf{U}_2{}^T & \mathbf{V}_2{}^T \end{bmatrix}$ is the displacement vector for nodes located in Ω_2 and $\mathbf{R}^T = \begin{bmatrix} \mathbf{R}_1{}^T & \mathbf{R}_2{}^T \end{bmatrix}$ is the total displacement vector. Substituting equations (3.1) and (6.22) in the variational principle (6.20)

$$\begin{aligned} \delta \mathbf{R}_1^T \mathbf{K}_1 \mathbf{R}_1 + \delta \mathbf{R}_2^T \mathbf{K}_2 \mathbf{R}_2 - \delta \mathbf{R}_1^T \mathbf{F}_{1t} - \delta \mathbf{R}_2^T \mathbf{F}_{2t} + \alpha \delta \mathbf{R}_1^T \mathbf{V}_1 \mathbf{R}_1 - \alpha \delta \mathbf{R}_1^T \mathbf{F}_{1u} + \\ \alpha \delta \mathbf{R}_2^T \mathbf{V}_2 \mathbf{R}_2 - \alpha \delta \mathbf{R}_2^T \mathbf{F}_{2u} + \delta \mathbf{R}_1^T \mathbf{F^c}_1 + \delta \mathbf{R}_2^T \mathbf{F^c}_2 = 0, \end{aligned} \tag{6.26}$$

where

$$\mathbf{K}_i = \int_{\Omega_i} \mathbf{B} \mathbf{D} \mathbf{B}^T d\Omega_i \qquad i = 1, 2, \tag{6.27}$$

where $\mathbf{D}$ is the stress-strain relationship matrix, $\mathbf{B}$ is the differential strain operator matrix

$$\mathbf{V}_i = \int_{\Gamma_{ui}} \phi \phi^T d\Gamma_{ui} \qquad i = 1, 2, \tag{6.28}$$

$$\mathbf{F}_{iu} = \int_{\Gamma_{ui}} \phi \bar{\mathbf{u}} d\Gamma_{ui} \qquad i = 1, 2, \tag{6.29}$$

$$\mathbf{F}_{it} = \int_{\Gamma_{ti}} \phi \bar{\mathbf{t}} d\Gamma_{ti} \qquad i = 1, 2, \tag{6.30}$$

and

$$\mathbf{F^c}_1 = \begin{bmatrix} \mathbf{F^c}_{x1} \\ \mathbf{F^c}_{y1} \end{bmatrix}, \tag{6.31}$$

$$\begin{bmatrix} \mathbf{F^c}_{x2} \\ \mathbf{F^c}_{y2} \end{bmatrix}, \tag{6.32}$$

$$\mathbf{F^c}_{x1} = \int_{\Gamma_c} \phi_{1c} \tau_t \mathrm{d}\Gamma_c, \tag{6.33}$$

$$\mathbf{F^c}_{y1} = \int_{\Gamma_c} \phi_{1c} \tau_n \mathrm{d}\Gamma_c, \tag{6.34}$$

$$\mathbf{F^c}_{x2} = -\int_{\Gamma_c} \phi_{2c} \tau_t \mathrm{d}\Gamma_c, \tag{6.35}$$

$$\mathbf{F^c}_{y2} = -\int_{\Gamma_c} \phi_{2c} \tau_n \mathrm{d}\Gamma_c. \tag{6.36}$$

Finally the following nonlinear set of equations can be obtained

$$\left[\mathbf{K} + \alpha \mathbf{V}\right] \mathbf{R} + \mathbf{F}^c(\mathbf{R}) - \mathbf{F} - \alpha \mathbf{F}_{\bar{u}} = \mathbf{f}(\mathbf{R}) = 0 \tag{6.37}$$

where

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_1 & 0 \\ 0 & \mathbf{K}_2 \end{bmatrix} \tag{6.38}$$

and

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & 0 \\ 0 & \mathbf{V}_2 \end{bmatrix}, \tag{6.39}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_{1t} \\ \mathbf{F}_{2t} \end{bmatrix}, \tag{6.40}$$

$$\mathbf{F}_{\bar{u}} = \begin{bmatrix} \mathbf{F}_{1u} \\ \mathbf{F}_{2u} \end{bmatrix}, \tag{6.41}$$

$$\mathbf{F}^c = \begin{bmatrix} \mathbf{F}^c_1 \\ \mathbf{F}^c_2 \end{bmatrix}. \tag{6.42}$$

6.3.3 Tangent Stiffness Matrix

Equation (6.37) must be solved iteratively. A commonly used scheme is the Newton-Raphson method where the iteration $n+1$ at a generic load step is obtained from iteration n by the formula

$$\mathbf{R}^{(n+1)} = \mathbf{R}^{(n)} - \left(\mathbf{J}^{(n)}\right)^{-1} \mathbf{f}(\mathbf{R}^{(n)}), \tag{6.43}$$

where

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{R}}. \tag{6.44}$$

In order to obtain the Jacobian matrix $\mathbf{J}$, a linearization of equation (6.21) is needed.

$$\tau^{(n+1)} = \tau^{(n)} + \frac{\partial \tau}{\partial \boldsymbol{\Delta}} \left(\boldsymbol{\Delta}^{(n+1)} - \boldsymbol{\Delta}^{(n)}\right) = \tau^{(n)} + \mathbf{T} \left(\boldsymbol{\Delta}^{(n+1)} - \boldsymbol{\Delta}^{(n)}\right), \tag{6.45}$$

where

$$\mathbf{T} = \frac{\partial \tau}{\partial \boldsymbol{\Delta}} = \begin{bmatrix} \dfrac{\partial K_x}{\partial \Delta_x} & 0 \\ 0 & \dfrac{\partial K_y}{\partial \Delta_y} \end{bmatrix} = \begin{bmatrix} -\dfrac{\partial D_x}{\partial \Delta_x} K_{0x} & 0 \\ 0 & -\dfrac{\partial D_y}{\partial \Delta_y} K_{0y} \end{bmatrix}. \tag{6.46}$$

Deriving equation (6.14)

$$\frac{\partial D}{\partial \Delta} = \frac{\Delta_f \Delta_0}{\Delta^2(\Delta_f - \Delta_0)}. \tag{6.47}$$

Substituting (6.46) into (6.20), the expression of the Jacobian (6.44) can be obtained as

$$\mathbf{J} = \mathbf{K} + \alpha \mathbf{V} + \mathbf{K}_T, \tag{6.48}$$

where

$$\mathbf{K}_T = \frac{\partial \mathbf{F^c}}{\partial \mathbf{R}} = \int_{\Gamma_c} \begin{bmatrix} \phi_{1c} & \\ & \phi_{1c} \\ -\phi_{2c} & \\ & -\phi_{2c} \end{bmatrix} \frac{\partial \tau}{\partial \boldsymbol{\Delta}} \frac{\partial \boldsymbol{\Delta}}{\partial \mathbf{R}} \mathrm{d}\Gamma_c. \tag{6.49}$$

Considering equations (6.25) and (6.46)

$$\mathbf{K}_T = \int_{\Gamma_c} \begin{bmatrix} \phi_{1c} & \\ & \phi_{1c} \\ -\phi_{2c} & \\ & -\phi_{2c} \end{bmatrix} \mathbf{T} \begin{bmatrix} \phi_{1c}^T & & -\phi_{2c}^T & \\ & \phi_{1c}^T & & -\phi_{2c}^T \end{bmatrix} \mathrm{d}\Gamma_c, \tag{6.50}$$

$$\mathbf{K}_T = \begin{bmatrix} \mathbf{K}_{T11} & -\mathbf{K}_{T12} \\ -\mathbf{K}_{T12}^T & \mathbf{K}_{T22} \end{bmatrix}, \tag{6.51}$$

where

$$\mathbf{K}_{T11} = \begin{bmatrix} \mathbf{K}_{T11x} & 0 \\ 0 & \mathbf{K}_{T11y} \end{bmatrix}, \tag{6.52}$$

$$\mathbf{K}_{T12} = \begin{bmatrix} \mathbf{K}_{T12x} & 0 \\ 0 & \mathbf{K}_{T12y} \end{bmatrix}, \tag{6.53}$$

$$\mathbf{K}_{T22} = \begin{bmatrix} \mathbf{K}_{T22x} & 0 \\ 0 & \mathbf{K}_{T22y} \end{bmatrix}, \tag{6.54}$$

$$\mathbf{K}_{Tijl} = \int_{\Gamma_c} \phi_{ic} \left(K_l(\Delta_l) - K_{l0} \frac{\Delta_f \Delta_0}{\Delta_l(\Delta_f - \Delta_0)} \right) \phi_{jc} d\Gamma_c \qquad i,j = 1,2 \qquad l = x,y. \tag{6.55}$$

The equilibrium path for equation (6.37) will be traced with a single parameter arc-length continuation method, whose algorithm is explained in appendix B. The continuation method is essentially based on the Newton-Rhapson algorithm, with some modification in order to take into account the addition of the parameter, which in this case is the magnitude of the applied force. Therefore the method used is a *force control* method. A drawback of the Newton-Rhapson algorithms is that their (quadratic) convergence is guaranteed only if the guess solution is close enough to the real solution. In order to accelerate the Newton process, a *cubic polynomial line search* (LS) algorithm has been used Press *et al.* (1986). LS algorithms are instead globally convergent, since they may converge independently from the tentative solution.

6.3.4 Convergence Issues in the Numerical Continuation

As reported in Camanho & Dávila (2002) and Camanho *et al.* (2003), the softening nature of the bilinear cohesive zone model can pose numerical problems. These problems are related to the convergence of the numerical scheme, that may not be achieved in some cases. In fact, during the numerical simulations, it has been experienced that the numerical continuation stops and does not proceed further. The point of arrest is usually the first *unstable* point encountered in the

path, which is when the delamination starts to propagate. In other words, this point occurs when the relative displacement exceeds the critical value δ_f, which means that the strain energy release rate is greater than the critical value G_c for that particular mode, therefore, according to the Griffith's theory, there is enough energy to create *new* surfaces (the crack surfaces).

The choice of the penalty values is important since it could lead to large unbalanced forces and shoot the iteration beyond its radius of convergence. Particularly, the value of inter-facial maximum strength τ_m exerts a major influence on the *fluency* of the numerical scheme. It has been reported in Turon (2007), for finite elements this effect is due to the length of the cohesive zone. The length of the cohesive zone is defined as the distance from the crack tip (when the traction is zero with the bilinear model) to the point where the maximum cohesive traction τ_m is attained. If the cohesive zone is discretized by too few elements, the distribution of tractions ahead of the crack tip is misrepresented, analogously to the under-sampling phenomenon of a digital signal.

Therefore a very fine mesh should be used if the cohesive length is too small. The cohesive length is usually proportional to the fracture energy release rate G_c and to $\frac{1}{\tau_m^2}$. The higher the inter-facial strength, the finer the mesh that should be used. This suggests a way to use coarser mesh, i.e. to arbitrarily reduce the inter-facial strength, as done in Chen *et al.* (1999) and Alfano & Crisfield (2001). This reduction, though, does not compromise the resulting force-displacement curve, if the reduction is *reasonable*, but can indeed ease the convergence. A strong reduction, on the contrary, can lead to inaccurate results; therefore the reduction of the τ_m should be used carefully.

Another problem related to the insufficient mesh size is the *false instabilities*. Although the critical value can be accurately predicted, an oscillatory behaviour can be observed in the force-displacement curve. In the finite element context, these oscillations in the force-displacement curve are due to a sudden release of the elastic strain energy caused by simultaneous failure of more than one element per time. Unfortunately, reducing the step-length size in the arc-length procedure does not solve this inconvenient, since it is inherently caused by insufficient discretization of the interface, i.e. not enough elements.

False instabilities will be observed in a later *ad-hoc* example when an artificial *void* is introduced in the cohesive contact. This cavity is introduced simply by setting the penalty parameter to zero in an internal zone of the interface. One remedy to this issue is obviously to refine (globally or locally). Within certain limits, though, refinement can be avoided by adding more nodes to the enrichment, although this increases the number of extra-unknowns.

6.3.5 The class `CohesiveContact`

6.3.5.1 The properties

The properties of the `CohesiveContact` are

- `d0t`, `dFt`, `d0n`, `dFn`, `GIc`, `GIIc`, `smax_I`, `smax_II` are the parameters of the cohesive zone model, as described in section 6.2;
- `tx` and `ty`: the tractions at the interface as defined in equation (6.21);
- `vx` and `vy`: the displacement jumps at the interface as in equation (6.21);
- `ne1` and `ne2`: the `ne1`-*th* edge of Ω_1 and the `ne2`-*th* edge of Ω_2 where the penalty contact is applied;
- `RKPM1` `RKPM2` are objects of the class `RKPM` that calculates ϕ_{1c} and ϕ_{2c} in equation (6.23) and (6.24);
- `hn`, `ht` arrays defined on the Gaussian points of the shared edge, with values 0 if there is a pre-crack, 1 otherwise;
- `KTt` and `KTn` the derivatives of the penalty factors, as defined in equation (6.55).

6.3.5.2 The methods

The methods defined for the class `CohesiveContact` are

- `CohesiveContact` the constructor, with inputs `ne1`, `ne2` and the file containing the CZM parameters;

- `ApplyContact` creates the objects `RKPM1` and `RKPM2`;
- `InitSegm` allows to insert a pre-crack by entering a logical expression like $x \geq L_x - a_0$;
- `DisplacementJump` calculates equation (6.22)

```
function obj = DisplacementJump(obj,U1,V1,U2,V2)
obj.vx = obj.RKPM1.PHI*U1-obj.RKPM2.PHI*U2;
obj.vy = obj.RKPM1.PHI*V1-obj.RKPM2.PHI*V2;
end
```

- `CohesiveTraction` calculates (6.21) and `KTt` and `KTn`

```
function obj = CohesiveTraction(obj,U1,V1,U2,V2)
obj = DisplacementJump(obj,U1,V1,U2,V2);
 [Dt,dDt] = dam(obj.vx,obj.d0t,obj.dFt);
 obj.kt = (1-Dt).*obj.Kt0.*obj.ht;
 obj.tx = obj.kt.*obj.vx;
obj.KTt = ((1-Dt) - obj.vx.*dDt).*obj.Kt0.*obj.ht;

[Dn,dDn] = dam(obj.vy,obj.d0n,obj.dFn);
obj.kn = (1-Dn).*obj.Kn0.*obj.hn;
obj.ty = obj.kn.*obj.vy;
obj.KTn = ((1-Dn) - obj.vy.*dDn).*obj.Kn0.*obj.hn;
clear Dt Dn dDn dDt
end
```

  where `dam` is an internal function that calculates (6.14);

- `AssembCoupl` computes the generalized cohesive force vector (6.42) and the tangent stiffness matrix (6.51) using the integration methods defined on the `RKPM` objects.

6.4 The Cohesive Segments method

The method of cohesive segments consists in a local partition of unity enrichment of an existing meshless approximation. It was developed initially in a FE framework Wells *et al.* (2002), Remmers *et al.* (2003b) and Remmers *et al.* (2003a) but using the PUM it can be extended to meshfree approximations.

In subsection 6.4.1 the approximation is described, then in subsection 6.4.2 the constructed approximation is used in a variational form in order to get the desired discretized equations of equilibrium.

Figure 6.9 shows the basic principles of cohesive segments. The idea is to decompose a continuous crack into *discrete* cracks, each one of them represented by a *segment*. On these segments a Heaviside enrichment function is defined. This enrichment is then superimposed to a continuous field that represents the *undamaged* model.

6.4.1 RKPM Enriched Approximation

The approximation $u^h(\mathbf{x})$ for a scalar variable is

$$\mathbf{u}^h(\mathbf{x}) = \bar{\mathbf{u}}(\mathbf{x}) + \sum_{j=1}^{m} \mathcal{H}_{\Gamma,j}(\mathbf{x})\tilde{\mathbf{u}}_j(\mathbf{x}), \tag{6.56}$$

where $\bar{\mathbf{u}}(\mathbf{x})$ is the *continuous* part of the displacement, m is the number of cracks, or cohesive segments, $\mathcal{H}_{\Gamma,j}$ is the enrichment function defined on the *j-th* segment and $\tilde{\mathbf{u}}_j(\mathbf{x})$ is a continuous field that defines the magnitude of the discontinuity.

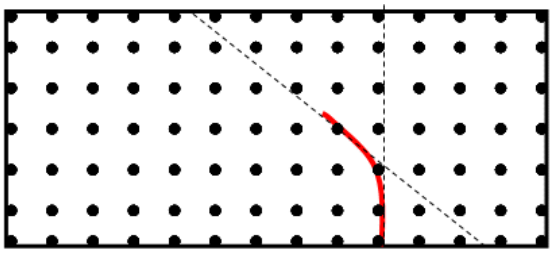


Figure 6.9: Cohesive segments method: circles: RKPM nodes; red line: crack; dashed line: cohesive segments

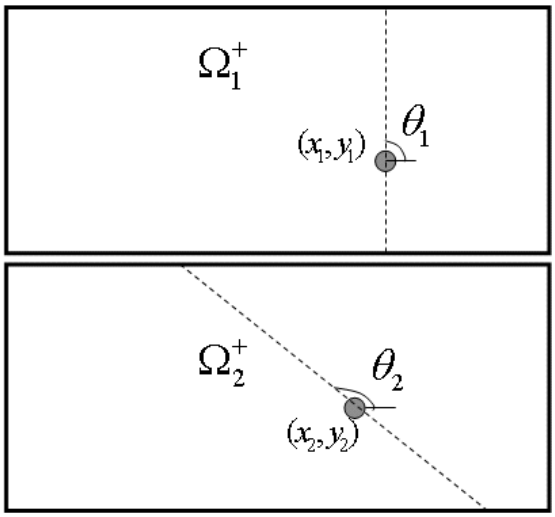


Figure 6.10: Decomposition of a continuous crack in discrete cracks

The enrichment function is a composed function of a Heaviside function and a distance function φ_j

$$\varphi_j(\mathbf{x}) = -\sin(\theta_j)(x - x_j) + \cos(\theta_j)(y - y_j), \tag{6.57}$$

where (x_j, y_j) and $\theta_j$are respectively the coordinates of the initial point and the inclination of the *j-th* segment, as illustrated in figure 6.10.

The normal vector of the segment (pointing towards Ω_j^+) is given by

$$\mathbf{n}_j^T = \begin{bmatrix} n_{xj} & n_{yj} \end{bmatrix} = \begin{bmatrix} \cos(\theta_j) & \sin(\theta_j) \end{bmatrix}. \tag{6.58}$$

The distance functions separate the domain Ω in two parts, as in figure 6.10.

$$\Omega_j^+ = \{\mathbf{x} \in \Omega : \varphi_j(\mathbf{x}) \geq 0\} \tag{6.59}$$

$$\Omega_j^- = \Omega - \Omega_j^+. \tag{6.60}$$

The cohesive segment is defined then as

$$\Gamma_{d,j} = \{\mathbf{x} \in \Omega : \varphi_j(\mathbf{x}) = 0\}. \tag{6.61}$$

Therefore, the enrichment functions are

$$\mathcal{H}_{\Gamma,j}(\mathbf{x}) = \mathcal{H}(\varphi_j(\mathbf{x})). \tag{6.62}$$

The infinitesimal strain can be calculated as follows

$$\epsilon^h = \nabla^s \bar{\mathbf{u}}(\mathbf{x}) + \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x})) \nabla^s \tilde{\mathbf{u}}_j(\mathbf{x}) + \delta(\varphi_j(\mathbf{x})) \left(\tilde{\mathbf{u}}_j(\mathbf{x}) \otimes \mathbf{n}_j\right), \tag{6.63}$$

where the superscript ^{s} means the symmetric part of the gradient $\nabla \bar{\mathbf{u}}(\mathbf{x})$ and the δ[1] is the Dirac function centered on the segment $\Gamma_{d,j}$[2], i.e. when $\varphi_j(\mathbf{x}) = 0$. It can be shown later that the *Dirac* term will originate the cohesive traction in the cohesive segment. This is opposed to the classic linear elastic fracture mechanics theory Rice (1968), where the fracture faces are considered traction-less, causing the Dirac term to disappear.

Using the Voigt notation, equation (4.96) can be rearranged as

$$\epsilon^h = \mathcal{L} \bar{\mathbf{u}}(\mathbf{x}) + \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x})) \mathcal{L} \tilde{\mathbf{u}}_j(\mathbf{x}) + \delta(\varphi_j(\mathbf{x})) \mathfrak{N} \tilde{\mathbf{u}}_j(\mathbf{x}), \tag{6.64}$$

where

$$\mathfrak{N} = \begin{bmatrix} n_{xj} & \\ & n_{yj} \\ n_{yj} & n_{xj}. \end{bmatrix} \tag{6.65}$$

The RKPM approximation is used for the continuous field $\bar{\mathbf{u}}(\mathbf{x})$ and $\tilde{\mathbf{u}}_j(\mathbf{x})$

$$u^h(\mathbf{x}) = \sum_{I=1}^{N} \phi_I(\mathbf{x}) U_I + \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x})) \sum_{i=1}^{N_j} \phi_i^{(0)}(\mathbf{x}) a_{ij} \quad \mathbf{x} \in \Omega, \tag{6.66}$$

where Ω is the domain of the elastic problem, N is the number of the *standard* nodes (not enriched), ϕ_I is the $RKPM$ shape function for the $I-th$ node, U_I is the $I-th$ (fictitious) displacement in node I , N_j is the number of the enriched nodes for the *j-th* crack, $\phi_i^{(0)}(\mathbf{x})$ is the *Shepard* shape functions for the $i-th$ enriched node, a_{ij} is the $i-th$ additional variable for the additional crack.

Similarly, for the v component of the displacement

$$v^h(\mathbf{x}) = \sum_{I=1}^{N} \phi_I(\mathbf{x}) V_I + \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x})) \sum_{i=1}^{N_j} \phi_i^{(0)}(\mathbf{x}) b_{ij} \quad \mathbf{x} \in \Omega. \tag{6.67}$$

[1] not be confused with the variational δ

[2] d means *discontinuity*

If $U_I = V_I = 0 \; \forall I$ and $a_{ij} = b_{ij} = 1 \; \forall J$, the functions (6.66) and (6.67) resort to a sum of Heaviside functions that represent the discontinuities.

Finally, the displacement jump at the *k-th* segment is given by

$$\boldsymbol{\Delta}_k(\mathbf{x}) = \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x}))\tilde{\mathbf{u}}_j(\mathbf{x}) = \\ = \sum_{j=1}^{m} \mathcal{H}(\varphi_j(\mathbf{x})) \begin{bmatrix} \sum_{i=1}^{N_j} \phi_i^{(0)}(\mathbf{x}) a_{ij} \\ \sum_{i=1}^{N_j} \phi_i^{(0)}(\mathbf{x}) b_{ij} \end{bmatrix} \qquad \mathbf{x} \in \Gamma_{d,k}. \quad (6.68)$$

6.4.2 The equations of equilibrium

The equations of equilibrium in strong form are the same of section 6.3, i.e. (6.15), (6.16), (6.17) with the exception of equation (6.18) that is modified to allow multiple segments

$$\mathbf{n}_j \cdot \sigma = \tau(\boldsymbol{\Delta}_{l,j}) \qquad \mathbf{x} \in \Gamma_{d,j} \qquad j = 1, \ldots, m, \quad (6.69)$$

where $\mathbf{n}_j$ is the normal unity vector of the segment $\Gamma_{d,j}$ where the cohesive traction $\tau(\boldsymbol{\Delta}_l)$ is prescribed, l is the local coordinate system.

The traction τ depends on the displacement jump $\boldsymbol{\Delta}_l$ (transformed in the local reference $t - n$ where t is the tangent to the segment and n is the normal direction to the segment) on the *cohesive segments* $\Gamma_{d,j}$ as described in section 6.2.

The displacement jump $\boldsymbol{\Delta}(\mathbf{x})_l$ in the local global reference for the *j-th* segment instead is

$$\boldsymbol{\Delta}_{l,j}(\mathbf{x}) = \mathbf{R}_j(\theta)\boldsymbol{\Delta}_j(\mathbf{x}) \qquad \mathbf{x} \in \Gamma_{d,j} \quad (6.70)$$

where

$$\mathbf{R}_j(\theta) = \begin{bmatrix} \cos(\theta_j) & \sin(\theta_j) \\ \\ -\sin(\theta_j) & \cos(\theta_j) \end{bmatrix}. \quad (6.71)$$

Using displacement $\mathbf{u}$ as test function for equations (6.15), (6.16) and the (local) displacement jump $\boldsymbol{\Delta}_l$ as test function for equation (6.18), the variational

principle can be written as

$$\int_{\Omega} \delta\epsilon^T \sigma d\Omega - \int_{\Gamma_t} \delta\mathbf{u}^T \bar{\mathbf{t}} d\Gamma_t + \frac{\alpha}{2}\delta \int_{\Gamma_u} (\mathbf{u} - \bar{\mathbf{u}})^2 d\Gamma_u + \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \delta\boldsymbol{\Delta}_{\mathbf{j}_l}^T \tau(\boldsymbol{\Delta}_{\mathbf{j}}) d\Gamma_{d,j} = 0, \tag{6.72}$$

where the penalty method is used to enforce essential boundary conditions (6.17).

In vectorial form equation (6.66) can be written as

$$u^h(\mathbf{x}) = \underbrace{\boldsymbol{\Phi}^T(\mathbf{x})}_{1 \text{ X } N} \underbrace{\mathbf{U}}_{N \text{ X } 1} + \sum_{j=1}^{m} \underbrace{\boldsymbol{\Phi}_{\mathbf{j}}^{\mathbf{0}T}(\mathbf{x})}_{1 \text{ X } N_j} \underbrace{\mathbf{a}_j}_{N_j \text{ X } 1} \tag{6.73}$$

where

$$\boldsymbol{\Phi}^T = \begin{bmatrix} \phi_1(\mathbf{x}) & \phi_2(\mathbf{x}) & \dots & \phi_N(\mathbf{x}) \end{bmatrix} \tag{6.74}$$

and similarly for

$$\boldsymbol{\Phi}_{\mathbf{j}}^{\mathbf{0}T} = \begin{bmatrix} \phi_1^{(0)}(\mathbf{x}) & \phi_2^{(0)}(\mathbf{x}) & \dots & \phi_{N_j}^{(0)}(\mathbf{x}) \end{bmatrix}. \tag{6.75}$$

Analogously for the v displacement

$$v^h(\mathbf{x}) = \underbrace{\boldsymbol{\Phi}^T(\mathbf{x})}_{1 \text{ X } N} \underbrace{\mathbf{V}}_{N \text{ X } 1} + \sum_{j=1}^{m} \underbrace{\boldsymbol{\Phi}_{\mathbf{j}}^{\mathbf{0}T}(\mathbf{x})}_{1 \text{ X } N_j} \underbrace{\mathbf{b}_j}_{N_j \text{ X } 1}. \tag{6.76}$$

Therefore, applying the differential operator as in equation (4.96)

$$\epsilon = \mathbf{B} \begin{bmatrix} \mathbf{U} \\ \\ \mathbf{V} \end{bmatrix} + \sum_{j=1}^{m} \mathcal{H}_j \mathbf{B}_j^{(0)} \begin{bmatrix} \mathbf{a_j} \\ \\ \mathbf{b_j} \end{bmatrix} \tag{6.77}$$

and, applying the generalized Hooke's law

$$\sigma = \mathbf{DB} \begin{bmatrix} \mathbf{U} \\ \\ \mathbf{V} \end{bmatrix} + \sum_{j=1}^{m} \mathcal{H}_j \mathbf{D}\mathbf{B}_j^{(0)} \begin{bmatrix} \mathbf{a_j} \\ \\ \mathbf{b_j} \end{bmatrix}, \tag{6.78}$$

the first term in (6.20) reads as

$$\int_\Omega \delta\epsilon^T \sigma \mathrm{d}\Omega = \int_\Omega \left(\delta\mathbf{R}^T\mathbf{B}^T + \sum_{j=1}^{m} \delta\mathbf{c}_j^T \mathcal{H}_j \mathbf{B}_j^{(0)T}\right) \mathbf{D} \left(\mathbf{B}\mathbf{R} + \sum_{k=1}^{m} \mathcal{H}_k \mathbf{B}_k^{(0)T} \mathbf{c}_k\right) \mathrm{d}\Omega =$$
$$= \delta\mathbf{R}^T \left[\int_\Omega \mathbf{B}^T\mathbf{D}\mathbf{B} d\Omega\right] \mathbf{R} + \delta\mathbf{R}^T \sum_{j=1}^{m} \left[\int_\Omega \mathbf{B}^T\mathbf{D}\mathbf{B}^{(0)T}\mathcal{H}_j d\Omega\right] \mathbf{c}_j +$$
$$+ \sum_{k=1}^{m} \delta\mathbf{c}_k^T \left[\int_\Omega \mathbf{B}^{(0)}\mathbf{D}\mathbf{B}\mathcal{H}_k d\Omega\right] \mathbf{R} + \sum_{k=1}^{m}\sum_{j=1}^{m} \delta\mathbf{c}_k^T \left[\int_\Omega \mathbf{B}^{(0)T}\mathbf{D}\mathbf{B}^{(0)}\mathcal{H}_k\mathcal{H}_j d\Omega\right] \mathbf{c}_j \tag{6.79}$$

where $\mathbf{R}^T = \begin{bmatrix}\mathbf{U}^T & , & \mathbf{V}^T\end{bmatrix}$ and $\mathbf{c}_j^T = \begin{bmatrix}\mathbf{a}_j^T & , & \mathbf{b}_j^T\end{bmatrix}$ and $\mathcal{H}_j$ is the enrichment function as in equation (6.62).

The variational term for the forces is as follow

$$\int_{\Gamma_t} \delta\mathbf{u}^T \bar{\mathbf{t}} \mathrm{d}\Gamma = \delta\mathbf{R}^T \begin{bmatrix}\int_{\Gamma_t} \mathbf{\Phi}\bar{t}_x \mathrm{d}\Gamma \\ \int_{\Gamma_t} \mathbf{\Phi}\bar{t}_y \mathrm{d}\Gamma\end{bmatrix} + \sum_{j=1}^{m} \delta\mathbf{c}_j^T \mathcal{H}_j \begin{bmatrix}\int_{\Gamma_t} \mathbf{\Phi^{(0)}}\bar{t}_x \mathrm{d}\Gamma \\ \int_{\Gamma_t} \mathbf{\Phi^{(0)}}\bar{t}_y \mathrm{d}\Gamma\end{bmatrix}. \tag{6.80}$$

The same term can be obtained for the cohesive forces, i.e. the term

$$\int_{\Gamma_{l,j}} \delta\mathbf{\Delta}_{l,j}^T \tau(\mathbf{\Delta}_{l,j}) \mathrm{d}\Gamma_{d,j}. \tag{6.81}$$

In a non-linear perspective, the relative displacement $\mathbf{\Delta}_{l,j}$ must be calculated first, in order to obtain the traction τ. Using the coordinate transformation (6.70),

$$\mathbf{\Delta}_{l,j} = \mathbf{R}_j(\theta)\mathbf{\Delta}_j = \mathbf{R}_j(\theta) \sum_{k=1}^{m} \mathcal{H}_k \begin{bmatrix}\mathbf{\Phi}^{\mathbf{T}(0)}_k \mathbf{a}_k \\ \mathbf{\Phi}^{\mathbf{T}(0)}_k \mathbf{b}_k\end{bmatrix} \qquad \mathbf{x} \in \Gamma_{d,j}, \tag{6.82}$$

$$\int_{\Gamma_{d,j}} \delta\mathbf{\Delta_{l,j}}^T \tau(\mathbf{\Delta_{l,j}}) \mathrm{d}\Gamma_{d,j} =$$
$$= \sum_{k=1}^{m} \delta\mathbf{c}_k^T \mathcal{H}_k \begin{bmatrix}\int_{\Gamma_{d,j}} \mathbf{\Phi^{(0)}}_k \left[\tau_t(\mathbf{\Delta_{l,j}})\cos(\theta_j) - \tau_n(\mathbf{\Delta_{l,j}})\sin(\theta_j)\right] \mathrm{d}\Gamma_{d,j} \\ \int_{\Gamma_{d,j}} \mathbf{\Phi^{(0)}}_k \left[\tau_t(\mathbf{\Delta_{l,j}})\sin(\theta_j) + \tau_n(\mathbf{\Delta_{l,j}})\cos(\theta_j)\right] \mathrm{d}\Gamma_{d,j}\end{bmatrix}. \tag{6.83}$$

Summing over $j = 1, \ldots, m$ as in equation (6.20)

$$\sum_{j=1}^{m} \int_{\Gamma_{d,j}} \delta \boldsymbol{\Delta}_{\mathbf{l,j}}{}^{T} \tau(\boldsymbol{\Delta}_{\mathbf{l,j}}) \mathrm{d}\Gamma_{d,j} =$$
$$= \sum_{j=1}^{m} \sum_{k=1}^{m} \delta \mathbf{c}_k^T \begin{bmatrix} \int_{\Gamma_{d,j}} \boldsymbol{\Phi}^{(\mathbf{0})}{}_k \left[\tau_t(\boldsymbol{\Delta}_{\mathbf{l,j}}) \cos(\theta_j) - \tau_n(\boldsymbol{\Delta}_{\mathbf{l,j}}) \sin(\theta_j) \right] \mathcal{H}_k \mathrm{d}\Gamma_{d,j} \\ \int_{\Gamma_{d,j}} \boldsymbol{\Phi}^{(\mathbf{0})}{}_k \left[\tau_t(\boldsymbol{\Delta}_{\mathbf{l,j}}) \sin(\theta_j) + \tau_n(\boldsymbol{\Delta}_{\mathbf{l,j}}) \cos(\theta_j) \right] \mathcal{H}_k \mathrm{d}\Gamma_{d,j} \end{bmatrix}. \quad (6.84)$$

Equation (6.84) can be regarded as a nonlinear *internal force* term. It should be remarked that the *integration* is carried out over the union of all the segments, regardless if they intersect or not

$$\Gamma_d = \bigcup_{j=1}^{m} \Gamma_{d,j}. \quad (6.85)$$

Another remark is that the (6.84) is similar to (6.80) since they both integrate tractions on boundaries and the result is a *force* vector in the final discretized equations.

6.4.3 Discretization of equations of equilibrium

The final equations from (6.20) are

$$\left(\begin{bmatrix} \mathbf{K} & \mathbf{K}_{rc_1} & \ldots & \mathbf{K}_{rc_m} \\ & \mathbf{K}_{c_1c_1} & \ldots & \mathbf{K}_{c_1c_m} \\ & & \ddots & \vdots \\ & \textit{symm.} & & \mathbf{K}_{c_m,c_m} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{V} & \mathbf{V}_{rc_1} & \ldots & \mathbf{V}_{rc_m} \\ & \mathbf{V}_{c_1c_1} & \ldots & \mathbf{V}_{c_1c_m} \\ & & \ddots & \vdots \\ & \textit{symm.} & & \mathbf{V}_{c_mc_m} \end{bmatrix} \right) \begin{bmatrix} \mathbf{R} \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{bmatrix} +$$
$$- \begin{bmatrix} \mathbf{F} \\ \mathbf{F}_{c_1} \\ \vdots \\ \mathbf{F}_{c_m} \end{bmatrix} - \alpha \begin{bmatrix} \mathbf{F_u} \\ \mathbf{F}_{\mathbf{u}c_1} \\ \vdots \\ \mathbf{F}_{\mathbf{u}c_m} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{F^c}_{c_1}(\mathbf{c}_1, \ldots, \mathbf{c}_m) \\ \vdots \\ \mathbf{F^c}_{c_m}(\mathbf{c}_1, \ldots, \mathbf{c}_m) \end{bmatrix} = 0 \quad (6.86)$$

where

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^T \mathbf{D} \mathbf{B} \mathrm{d}\Omega, \tag{6.87}$$

$$\mathbf{K}_{rc_j} = \int_{\Omega_j^+} \mathbf{B}^T \mathbf{D} \mathbf{B}_j^{(0)} \mathrm{d}\Omega, \tag{6.88}$$

$$\mathbf{K}_{c_i c_j} = \int_{\Omega_i^+ \cap \Omega_j^+} \mathbf{B}_i^{(0)T} \mathbf{D} \mathbf{B}_j^{(0)} \mathrm{d}\Omega, \tag{6.89}$$

$$\mathbf{V}_u = \mathbf{V}_v = \int_{\Gamma_u} \mathbf{\Phi} \mathbf{\Phi}^T \mathrm{d}\Gamma_u, \tag{6.90}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_u & \\ & \mathbf{V}_v \end{bmatrix}, \tag{6.91}$$

$$\mathbf{V}_{ua_j} = \mathbf{V}_{vb_j} = \int_{\Gamma_u} \mathbf{\Phi} \mathbf{\Phi}_j^{(0)T} \mathcal{H}_j \mathrm{d}\Gamma_u, \tag{6.92}$$

$$\mathbf{V}_{rc_j} = \begin{bmatrix} \mathbf{V}_{ua_j} & \\ & \mathbf{V}_{vb_j} \end{bmatrix}, \tag{6.93}$$

$$\mathbf{V}_{a_i a_j} = \mathbf{V}_{b_i b_j} = \int_{\Gamma_u} \mathbf{\Phi}_i^{(0)} \mathbf{\Phi}_j^{(0)T} \mathcal{H}_i \mathcal{H}_j \mathrm{d}\Gamma_u, \tag{6.94}$$

$$\mathbf{V}_{c_i c_j} = \begin{bmatrix} \mathbf{V}_{a_i a_j} & \\ & \mathbf{V}_{b_i b_j} \end{bmatrix}, \tag{6.95}$$

$$\mathbf{F}_x = \int_{\Gamma_t} \mathbf{\Phi} \bar{t}_x \mathrm{d}\Gamma_t, \tag{6.96}$$

$$\mathbf{F}_y = \int_{\Gamma_t} \mathbf{\Phi} \bar{t}_y \mathrm{d}\Gamma_t, \tag{6.97}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x \\ \mathbf{F}_y \end{bmatrix}, \tag{6.98}$$

$$\mathbf{F}_{x,c_i} = \int_{\Gamma_t} \mathbf{\Phi}_{\mathbf{i}}^{(\mathbf{0})} \bar{t}_x \mathcal{H}_i \mathrm{d}\Gamma_t, \tag{6.99}$$

$$\mathbf{F}_{y,c_i} = \int_{\Gamma_t} \mathbf{\Phi_i^{(0)}} \bar{t}_y \mathcal{H}_i \mathrm{d}\Gamma_t, \tag{6.100}$$

$$\mathbf{F}_{c_i} = \begin{bmatrix} \mathbf{F}_{x,c_i} \\ \\ \mathbf{F}_{y,c_i} \end{bmatrix}, \tag{6.101}$$

$$\mathbf{F}_{\mathbf{u}x} = \int_{\Gamma_t} \mathbf{\Phi} \bar{u} \mathrm{d}\Gamma_t, \tag{6.102}$$

$$\mathbf{F}_{\mathbf{u}y} = \int_{\Gamma_t} \mathbf{\Phi} \bar{v} \mathrm{d}\Gamma_t, \tag{6.103}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_{\mathbf{u}x} \\ \\ \mathbf{F}_{\mathbf{u}y} \end{bmatrix}, \tag{6.104}$$

$$\mathbf{F}_{\mathbf{u}x,c_i} = \int_{\Gamma_t} \mathbf{\Phi_i^{(0)}} \mathcal{H}_i \bar{u} \mathrm{d}\Gamma_t, \tag{6.105}$$

$$\mathbf{F}_{\mathbf{u}y,c_i} = \int_{\Gamma_t} \mathbf{\Phi_i^{(0)}} \mathcal{H}_i \bar{v} \mathrm{d}\Gamma_t, \tag{6.106}$$

$$\mathbf{F}_{\mathbf{u}c_i} = \begin{bmatrix} \mathbf{F}_{\mathbf{u}x,c_i} \\ \\ \mathbf{F}_{\mathbf{u}y,c_i} \end{bmatrix}, \tag{6.107}$$

$$\mathbf{F^c}_{x,c_i} = \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \mathbf{\Phi_i^{(0)}} \left[\tau_t(\mathbf{\Delta_{l,j}}) \cos(\theta_j) - \tau_n(\mathbf{\Delta_{l,j}}) \sin(\theta_j) \right] \mathcal{H}_i \mathrm{d}\Gamma_{d,j}, \tag{6.108}$$

$$\mathbf{F^c}_{y,c_i} = \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \mathbf{\Phi_i^{(0)}} \left[\tau_t(\mathbf{\Delta_{l,j}}) \sin(\theta_j) + \tau_n(\mathbf{\Delta_{l,j}}) \cos(\theta_j) \right] \mathcal{H}_i \mathrm{d}\Gamma_{d,j}, \tag{6.109}$$

$$\mathbf{F^c}_{c_i} = \begin{bmatrix} \mathbf{F^c}_{x,c_i} \\ \\ \mathbf{F^c}_{y,c_i} \end{bmatrix}. \tag{6.110}$$

6.4.4 Tangent Stiffness Matrix

Equation (6.86) is a nonlinear system of equations in the variable $\left[\mathbf{R}^T \quad , \quad \mathbf{c}^T\right]^T$. The solution to equation (6.86) is usually solved with an iterative scheme, often *Newton-Rhapson* (NR) and its modifications. NR method is based on a first order Taylor expansion with respect to the unknown $\left[\mathbf{R^T} \quad \mathbf{c^T}\right]^T$, therefore it requires the Jacobian $\mathbf{J}$ of the right hand side of the equation (6.86). This Jacobian is known as the *tangent stiffness matrix* and for equation (6.86) is

$$\begin{bmatrix} \mathbf{K} & \mathbf{K}_{rc_1} & \dots & \mathbf{K}_{rc_m} \\ & \mathbf{K}_{c_1c_1} & \dots & \mathbf{K}_{c_1c_m} \\ & & \ddots & \vdots \\ & \textit{symm.} & & \mathbf{K}_{c_m,c_m} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{V} & \mathbf{V}_{rc_1} & \dots & \mathbf{V}_{rc_m} \\ & \mathbf{V}_{c_1c_1} & \dots & \mathbf{V}_{c_1c_m} \\ & & \ddots & \vdots \\ & \textit{symm.} & & \mathbf{V}_{c_mc_m} \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 \\ & \mathbf{K_T}_{c_1c_1} & \dots & \mathbf{K_T}_{c_1c_m} \\ & & \ddots & \vdots \\ & \textit{symm.} & & \mathbf{K_T}_{c_m,c_m} \end{bmatrix} \tag{6.111}$$

where $\mathbf{K}_T$ can be obtained by partial derivation of equation (6.84) with respect to $\mathbf{c}_i$.

Indeed

$$\begin{aligned} \mathbf{K_T}_{ij} = \frac{\partial \mathbf{F^c}_{c,i}}{\partial \mathbf{c}_j} = \frac{\partial}{\partial \mathbf{c}_j} \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \begin{bmatrix} \mathbf{\Phi_i}(\mathbf{0})^T & & \\ & & \\ & & \mathbf{\Phi_i^{(0)}}^T \end{bmatrix} \mathbf{R}_j(\theta)^T \tau \ \mathcal{H}_i \mathrm{d}\Gamma_{d,j} \\ \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \begin{bmatrix} \mathbf{\Phi_i}(\mathbf{0})^T & & \\ & & \\ & & \mathbf{\Phi_i^{(0)}}^T \end{bmatrix} \mathbf{R}_j(\theta)^T \frac{\partial \tau}{\partial \mathbf{c}_j} \ \mathcal{H}_i \mathrm{d}\Gamma_{d,j} = \\ = \sum_{j=1}^{m} \int_{\Gamma_{d,j}} \begin{bmatrix} \mathbf{\Phi_i}(\mathbf{0})^T & & \\ & & \\ & & \mathbf{\Phi_i^{(0)}}^T \end{bmatrix} \mathbf{R}_j(\theta)^T \frac{\partial \tau}{\partial \mathbf{\Delta_{l,j}}} \frac{\partial \mathbf{\Delta_{l,j}}}{\partial \mathbf{c}_j} \ \mathcal{H}_i \mathrm{d}\Gamma_{d,j}. \end{aligned} \tag{6.112}$$

From equation (6.82)

$$\frac{\partial \boldsymbol{\Delta}_{l,j}}{\partial \mathbf{c}_j} = \mathbf{R}_j(\theta)\mathcal{H}_j \begin{bmatrix} \boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} & \\ & \boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \end{bmatrix}. \tag{6.113}$$

The Jacobian of the tractions with respect to the relative displacements is equation (6.46). Therefore,

$$\mathbf{K}_{\mathbf{T}ij} = \int_{\Gamma_{d,j}} \mathcal{H}_i\mathcal{H}_j \begin{bmatrix} \boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})} & \\ & \boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})} \end{bmatrix} \mathbf{R}_j(\theta)^T\mathbf{T}\mathbf{R}_j(\theta) \begin{bmatrix} \boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} & \\ & \boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \end{bmatrix} \mathrm{d}\Gamma_{d,j} = \begin{bmatrix} \mathbf{K}_{\mathbf{T}ij11} & \mathbf{K}_{\mathbf{T}ij12} \\ \mathbf{K}_{\mathbf{T}ij21} & \mathbf{K}_{\mathbf{T}ij22} \end{bmatrix} \tag{6.114}$$

where

$$\mathbf{K}_{\mathbf{T}ij11} = \int_{\Gamma_{d,j}} \mathcal{H}_i\mathcal{H}_j\boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})}\boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \cos(\theta_j)(\mathbf{T}_{11}\cos(\theta_j) - \mathbf{T}_{21}\sin(\theta_j)) + \\ - \sin(\theta_j)(\mathbf{T}_{12}\cos(\theta_j) - \mathbf{T}_{22}\sin(\theta_j))\mathrm{d}\Gamma_{d,j}, \tag{6.115}$$

$$\mathbf{K}_{\mathbf{T}ij12} = \int_{\Gamma_{d,j}} \mathcal{H}_i\mathcal{H}_j\boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})}\boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \cos(\theta_j)(\mathbf{T}_{12}\cos(\theta_j) - \mathbf{T}_{22}\sin(\theta_j)) + \\ + \sin(\theta_j)(\mathbf{T}_{11}\cos(\theta_j) - \mathbf{T}_{21}\sin(\theta_j))\mathrm{d}\Gamma_{d,j}, \tag{6.116}$$

$$\mathbf{K}_{\mathbf{T}ij21} = \int_{\Gamma_{d,j}} \mathcal{H}_i\mathcal{H}_j\boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})}\boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \cos(\theta_j)(\mathbf{T}_{21}\cos(\theta_j) + \mathbf{T}_{11}\sin(\theta_j)) + \\ - \sin(\theta_j)(\mathbf{T}_{222}\cos(\theta_j) + \mathbf{T}_{12}\sin(\theta_j))\mathrm{d}\Gamma_{d,j}, \tag{6.117}$$

$$\mathbf{K}_{\mathbf{T}ij22} = \int_{\Gamma_{d,j}} \mathcal{H}_i\mathcal{H}_j\boldsymbol{\Phi}_{\mathbf{i}}^{(\mathbf{0})}\boldsymbol{\Phi}_{\mathbf{j}}^{(\mathbf{0})^T} \cos(\theta_j)(\mathbf{T}_{22}\cos(\theta_j) + \mathbf{T}_{12}\sin(\theta_j)) + \\ + \sin(\theta_j)(\mathbf{T}_{21}\cos(\theta_j) + \mathbf{T}_{11}\sin(\theta_j))\mathrm{d}\Gamma_{d,j}. \tag{6.118}$$

6.4.5 Selection of Enriched Nodes

Equations (6.66) and (6.67) allow the introduction of the enrichment. The choice of the enriched nodes is therefore an important part of the algorithm, since it allows to *locally* introducing the discontinuity. Equations (6.66) and (6.67) are only valid if the PU is built on nodes whose support is cut (partially or entirely) by the discontinuity.

The selection is then made upon comparison of the distances between the node and the *segment* (figures 6.11 and 6.12). If the distance is less than the *dilatation* parameter, then the node is selected for enrichment, otherwise is discarded. Calling $S_{enr,j}$ the set of enriched nodes for the $j-th$ crack

$$S_{enr,j} = \{\mathbf{x}_I \in S_1 : |\mathbf{x}_I - \mathbf{x}| \leq \rho_I \qquad \mathbf{x} \in \Gamma_{d,j}\} \tag{6.119}$$

where S_1 is the set of RKPM nodes.

Naming $h(\mathbf{x}_I, \mathbf{x}) = |\mathbf{x}_I - \mathbf{x}|$ a distance function, equation (6.119) can be formulated as

$$S_{enr,j} = \{\mathbf{x}_I \in S_1 : h(\mathbf{x}_I, \mathbf{x}) \leq \rho_I \qquad \mathbf{x} \in \Gamma_{d,j}\}\,. \tag{6.120}$$

Hence, to correctly select the nodes, it is important to use an appropriate distance function. In two dimensions, if $\Gamma_{d,j}$ is a straight line, then it is easy to construct $h(\mathbf{x})$. If $\Gamma_{d,j}$ is instead a *segment* $\overline{P_1P_2}$

$$P_1 \equiv (x_1, y_1) \qquad P_2 \equiv (x_2, y_2) \tag{6.121}$$

the distance function is slightly more elaborated. In fact, it should take into account the endpoints of the segment, where the distance function is not the usual distance from a straight line, but it is the distance from a point. An appropriate distance function in this sense should blend these two distance function. In this section a quite easy approach is proposed. The basic idea is to use Heaviside functions on the parameter t that defines a segment

$$\begin{cases} x(t) = x_1 + t(x_2 - x_1) \\ y(t) = y_1 + t(y_2 - y_1) \qquad\qquad t \in [0,1] \end{cases}. \tag{6.122}$$

Defining

$$\mathbf{d}^T = \begin{bmatrix} x_2 - x_1 & , & y_2 - y_1 \end{bmatrix}, \tag{6.123}$$

$$\mathbf{r}_1^T = \begin{bmatrix} x - x_1 & , & y - y_1 \end{bmatrix}, \tag{6.124}$$

$$\mathbf{r}_2^T = \begin{bmatrix} x - x_2 & , & y - y_2 \end{bmatrix}, \tag{6.125}$$

then the parameter t can be defined $\forall\ (x, y) \in \mathrm{R}^2$

$$t = t(x, y) = \frac{\mathbf{d}^T \mathbf{r}_1}{\|\mathbf{d}\|^2} = \frac{(x_2 - x_1)(x - x_1) + (y_2 - y_1)(y - y_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{6.126}$$

then

$$a = (x - x_1)^2 + (y - y_1)^2, \tag{6.127}$$

$$b = (x - x_2)^2 + (y - y_2)^2, \tag{6.128}$$

$$c = [x - x_1 - t(x_2 - x_1)]^2 + [y - y_1 - t(y_2 - y_1)]^2. \tag{6.129}$$

Then the (minimum) distance function (not signed) $h(x, y)$ for a generic point (x, y) from the segment $\overline{P_1 P_2}$ is

$$h(x, y) = a(x, y)\mathcal{H}(-t(x, y)) + c(x, y)\left[\mathcal{H}(t(x, y)) - \mathcal{H}(t(x, y) - 1)\right] + \\ + b(x, y)\mathcal{H}(t(x, y) - 1), \tag{6.130}$$

where $\mathcal{H}$ is the *Heaviside* function.

The `Matlab` implementation can be easily done using *function handles* or *inline functions*. For example

```
t = inline('[(x2-x1).*(x-x1)+(y2-y1).*(y-y1)]./(((x2-x1).^2...
+(y2-y1).^2))','x','y','x1','x2','y1','y2');

a = inline('(x-x1).^2+(y-y1).^2','x','y','x1','y1');
```

```
b = inline('(x-x2).^2+(y-y2).^2','x','y','x2','y2');

c = inline('(x-(x1+t(x,y,x1,x2,y1,y2)*(x2-x1))).^2+...
...+(y-(y1+t(x,y,x1,x2,y1,y2)*(y2-y1))).^2'...
...,'x','y','x1','x2','y1','y2','t');

h = inline('a(x,y,x1,y1).*heaviside(-t(x,y,x1,x2,y1,y2))+...
...+c(x,y,x1,x2,y1,y2,t).*[heaviside(t(x,y,x1,x2,y1,y2))+...
-heaviside(t(x,y,x1,x2,y1,y2)-1)]+...
...+b(x,y,x2,y2).*heaviside(t(x,y,x1,x2,y1,y2)-1)',...
'x','y','x1','x2','y1','y2','a','b','c','t')
```

Once known the vectors $\mathbf{d}$ and $\mathbf{r}_1$, function h can be directly evaluated on the coordinates of the RKPM nodes and then compared to their respective dilatation parameters.

Figures 6.13 and 6.14 show the selection of the nodes using the function $h(x, y)$ in equation (6.130) for a segment shown in figure 6.11.

Figure 6.11: Cohesive segments: Gaussian points defined over the segment; red crosses: Gaussian points; black points: RKPM nodes

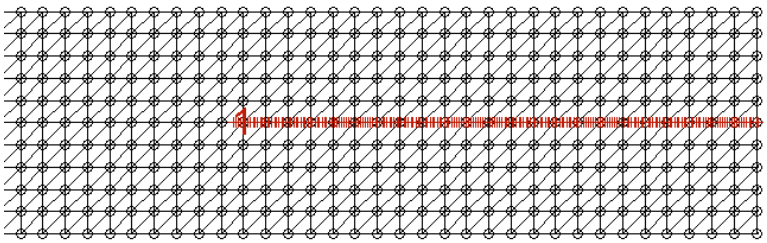


Figure 6.12: Cohesive segments: Gaussian points defined over the segment (zoom); red crosses: Gaussian points; black points: RKPM nodes

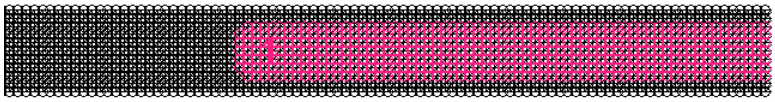


Figure 6.13: Cohesive segments: selection of enriched nodes; black: RKPM nodes; pink: enriched nodes

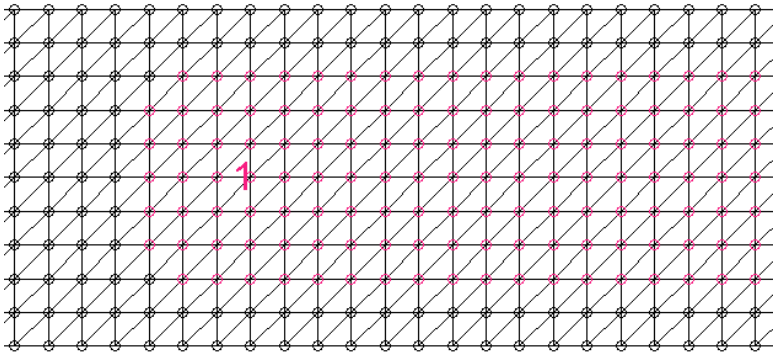


Figure 6.14: Cohesive segments: selection of enriched nodes (zoom); black: RKPM nodes; pink: enriched nodes

6.4.6 Numerical Examples

Numerical applications regarding single mode delamination are presented in this section. The fracture modes studied are mode I with three Double Cantilevered Beam (DCB) tests (figures 6.15 and 6.18) and mode II with an End Loaded Split test (ELS) (figure 6.16) and two End Notched Flexure (ENF) tests (figure 6.17). These tests were performed assuming that an initial crack already existed. In order to simulate an initial delamination (indicated with a_0), penalty factors were set to zero at the pre-crack.

An additional test (figure 6.19) with no initial crack was performed, with the scope of simulating initiation of the fracture.

Furthermore, a DCB beam with multiple failures is simulated to better understand the phenomena of *false instabilities* in the equilibrium path.

In all cases a full quadratic polynomial basis and regular arrangement of nodes have been used. By regular arrangement of nodes it is intended a distribution of nodes disposed as the vertices of a rectangular grid, as in figure 6.9. In the

following examples it will be indicated with $n_y \times n_x$, where n_y is the number of nodes along the thickness and n_x the number of nodes along the length.

The number n_x is chosen according to the aspect ratio of the plate L/b

$$n_x = \frac{L}{b} n_y \tag{6.131}$$

approximated to the next integer number.

The numerical quadrature used to calculate the stiffness matrices and the force vectors is Gaussian quadrature carried out over triangular elements, with a $3rd$ order quadrature.

The quadrature cells over the segments should be always constructed bearing in mind the under-sampling problem due to the cohesive zone. In the following examples the length is chosen to assure that the longest segment is discretized with at least 200 quadrature cells.

Although quadrilateral elements may seem more suitable for a regular arrangement of nodes. Nevertheless, triangular elements are preferred, since an automatic triangular mesh can always be constructed with ease; therefore the code gains in versatility and robustness. Moreover, since the kernel used is a circular support one, the advantage of a background mesh aligned with the support, as in Dolbow & Belytschko (1999) would be lost anyway. Nonetheless, accurate results can still be obtained, as shown in the next examples. The order of quadrature chosen is the best compromise between number of Gaussian points and quadrature. In fact, a higher order of quadrature could be more accurate, but leads to a much greater number of points. Since a $n-th$ order Gaussian quadrature rule has accuracy $\mathcal{O}(2n-1)$, for $n=3$ the accuracy is guaranteed up to the $5-th$ order.

A full quadratic basis was used for the standard RKPM approximation, while the enrichments have been constructed with Shepard functions, that are very easy to construct, as smooth as the kernel functions and have the same order of continuity. Furthermore, the `PSI`, `ig`, `js`, `dPSIcsi` and `dPSIeta` used for the RKPM can be stored and re-used to calculate the Shepard functions, allowing savings in computational run-time.

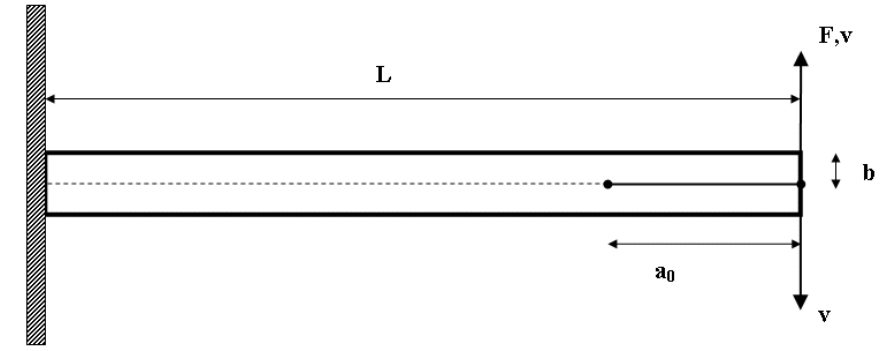


Figure 6.15: Double Cantilevered Beam: geometry and boundary conditions

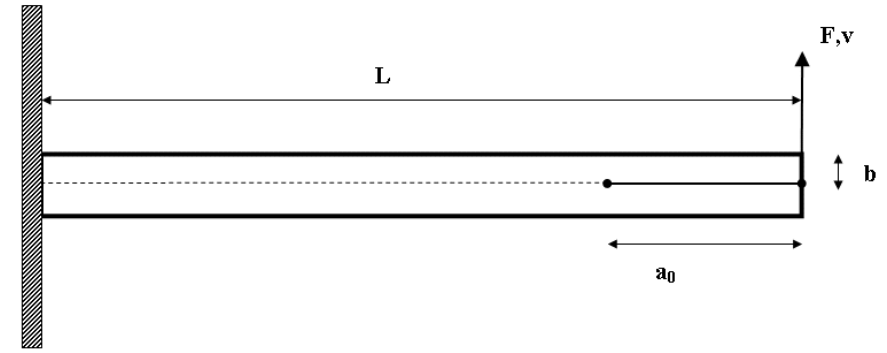


Figure 6.16: End Loaded Split: geometry and boundary conditions

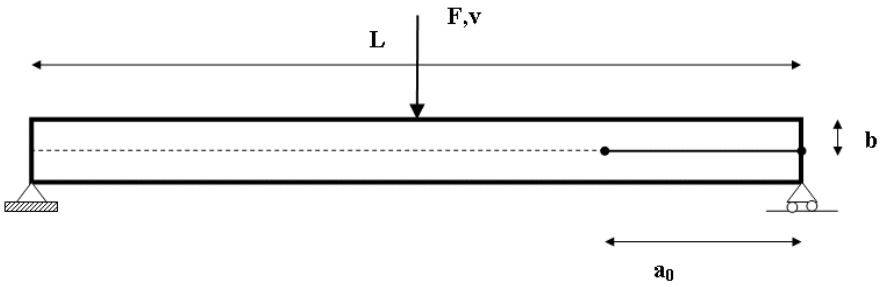


Figure 6.17: End Notched Flexure: geometry and boundary conditions

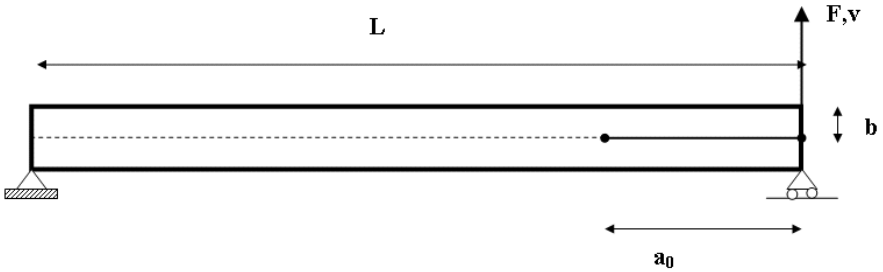


Figure 6.18: Simply Supported Double Cantilevered Beam: geometry and boundary conditions

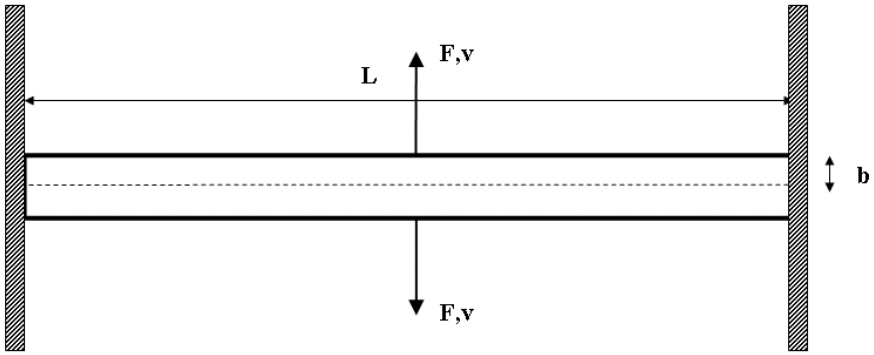


Figure 6.19: Central Delamination: geometry and boundary conditions

6.4.6.1 Double Cantilevered Beam: T300/977-2

The first case is a DCB test from Turon (2007) for a $(0°_{24})$ T300/977-2 carbon fibre-reinforced epoxy laminate. The elastic and cohesive properties and the geometry are resumed in table 6.1 (where w is the plate width).

E_{11}	E_{22}	G_{12}	ν_{12}
150 GPa	11 GPa	6 GPa	0.25

(a) Elastic Properties

L	b	w
150 mm	1.98 mm	20 mm

(b) Geometry for DCB test

L	b	w
100 mm	1.98 mm	10 mm

(c) Geometry for ENF test

G_{Ic}	τ_n^{max}	a_0
352 J/m^2	40 MPa	55 mm

(d) Mode I interface properties

G_{IIc}	τ_t^{max}	a_0
1450 J/m^2	40 MPa	30 mm

(e) Mode II interface properties

Table 6.1: Properties for T300/977-2

A comparison with the experimental test is shown in figure 6.21, as it can be observed the numerical model can predict the relative displacement at which the delamination starts (around 5 mm) and the correspondent maximum force (62 N). Once reached the critical opening displacement, the delamination propagates through the mid-plane. Figure (figure 6.20) shows different delamination stages. The deformed configuration is drawn in true scale. The *colormap* plot indicates different levels of transverse stress σ_y. As the delamination progresses, the stress concentration at the tip is tracked. The cohesive length zone could be observed in front of the crack tip, as the zone where the stress is more localized. The equilibrium path is fairly smooth and fits quite well the experimental tests. The *smoothness* derives from a quite fine distribution of nodes (a fine *mesh* using FE terminology) has been used, i.e. 10×758 nodes. This discretization is then able to capture the cohesive process.

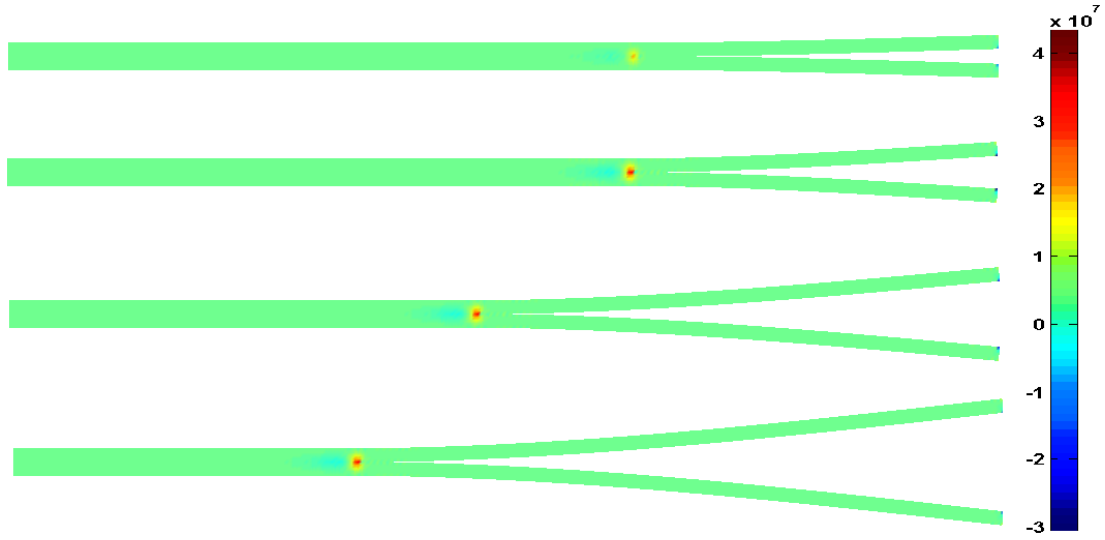


Figure 6.20: Delamination Stages for DCB test T300/977-2: transverse stress σ_y plot

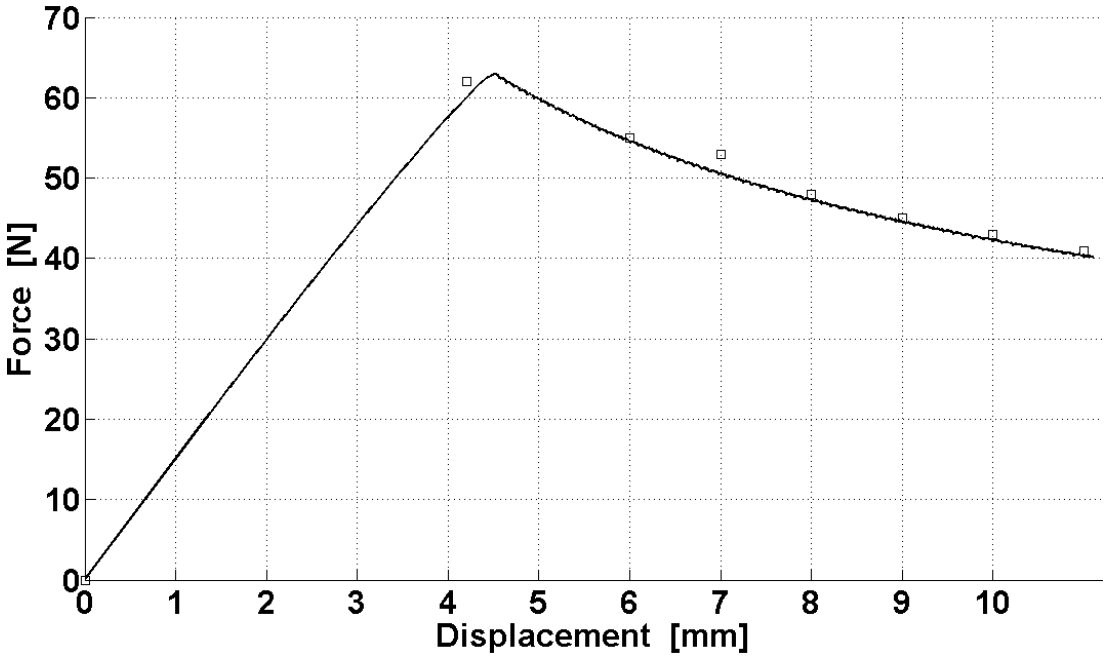


Figure 6.21: Force-Displacement curve for DCB test T300/977-2 : continuous line: numerical; squared line: experimental

6.4.6.2 Double Cantilevered Beam: XAS-913C

The second DCB case is from Alfano & Crisfield (2001) for a XAS-913C carbon-fiber epoxy composite. The distribution of nodes is 8×533. The properties and the initial conditions are reported in table 6.2. The plate has been modelled as figure 6.15 and there is only a slight difference between the experimental test and the numerical results regarding the reaction force (figure 6.22).

E_{11}	E_{22}	G_{12}	ν_{12}
126 GPa	7.5 GPa	4.981 GPa	0.281

(a) Elastic Properties

G_{Ic}	τ_m	a_0
263 J/m^2	57 MPa	30 mm

(b) Interface properties

L	b	w
100 mm	1.5mm	30 mm

(c) Geometry

Table 6.2: Properties for DCB test XAS-913C

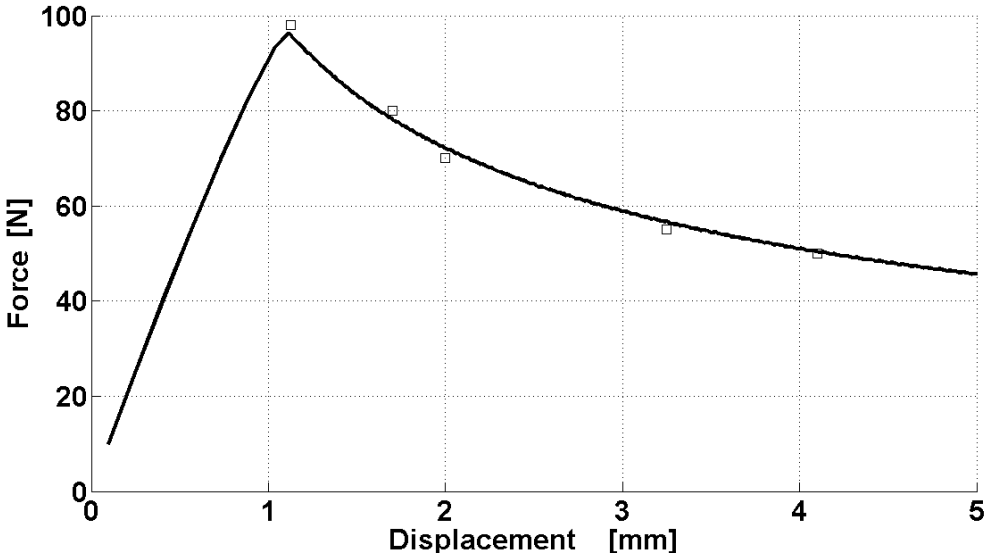
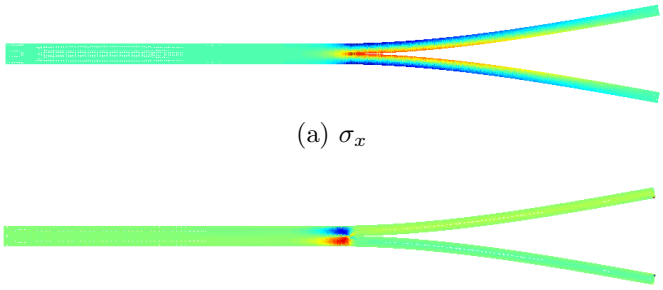


Figure 6.22: Force-Displacement curve for DCB test XAS-913C : continuous line: numerical; squared line: experimental

In figures 6.23 are depicted the stress color plot for the longitudinal stress

σ_x (figure 6.23a) and the shear stress τ_{xy} (figure 6.23b). As expected, each *delaminated* arm behaves like a cantilevered beam in bending constrained at the delamination tip. The same distribution can be identified in the next examples.



(a) σ_x

(b) τ_{xy}

Figure 6.23: Delamination stress plot for DCB test XAS-913C

6.4.6.3 Double Cantilevered Beam: AS4/PEEK

The third DCB case is taken from Camanho & Dávila (2002) for a thermoplastic laminate AS4/PEEK. The boundary conditions are in this case changed, as in figure 6.18. The plate is simply supported at both ends, loaded with a vertical force at the right end. Nevertheless, the plate is still loading in mode I. In fact, the only reaction force is located at the right end, opposite the external force, as in a DCB test. The elastic and interface properties are resumed in table 6.3 and the distribution of nodes is 8×523.

Even though local bumps in the equilibrium path appear to be more evident (figure 6.24), still there is a good fit between the numerical results and the experimental tests.

6.4.6.4 Double Cantilevered Beam: Plane Stress or Plane Strain?

In the next example a comparison between two different tensional states is presented. For real structures, there are two cases where the stress (or the strain) tensor can be simplified and some components neglected and therefore the analysis can be reduced from a three-dimensional stress analysis to a more simple and less expensive two-dimensional analysis. Such *two*-dimensional structure can

E_{11}	E_{22}	G_{12}	ν_{12}
122.7 GPa	10.1 GPa	5.5 GPa	0.25

(a) Elastic Properties

L	b	w
102 mm	1.56 mm	25.4 mm

(b) Geometry for DCB test

L	b	w
100 mm	1.56 mm	25.4 mm

(c) Geometry for ENF test

G_{Ic}	τ_n^{max}	a_0
969 J/m^2	80 MPa	32.9 mm

(d) Mode I interface properties

G_{IIc}	τ_t^{max}	a_0
1719 J/m^2	100 MPa	39.3 mm

(e) Mode II interface properties

Table 6.3: Properties for AS4/PEEK

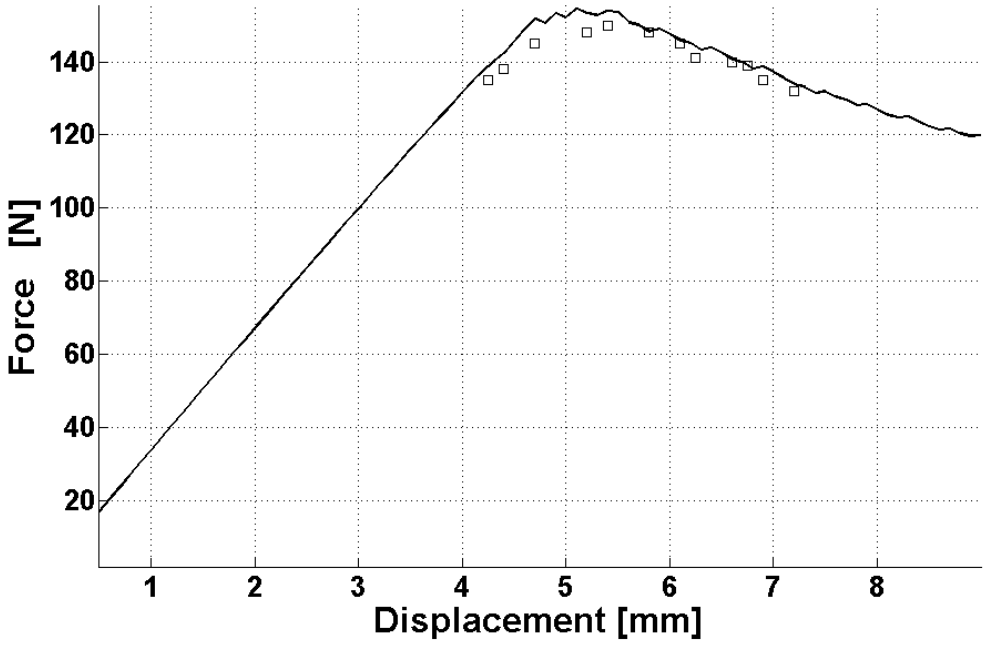


Figure 6.24: Force-Displacement curve for DCB test AS4/PEEK : continuous line: numerical; squared line: experimental

be either in *plane stress* or *plane strain*. A plane stress state is usually found in structures with one dimension much smaller than the other two, while for structures with one dimension much larger than the other two a plane strain state can be more properly assumed.

The link between the two theories is equation (4.99) and one can pass from one analysis to the other just changing the elastic properties. Nevertheless, the stress states are rather different. Supposing z the third dimension, a plane stress state assumes that $\sigma_{zz} \approx 0$, but according to Hooke's law, the consequence is that $\epsilon_{zz} \neq 0$, while for plane strain it is assumed that $\epsilon_{zz} \approx 0$ with a consequent $\sigma_{zz} \neq 0$. Both these non-zero terms, though, can be temporarily removed from the analysis and reducing the problem to a two dimensional one.

In the case of delamination, the length L is usually the largest dimension, but also the dimension where the delamination propagates, so it cannot be neglected. The dimension that can be neglected is the width w, since the relevant mechanisms for delamination occur in the length and in the thickness $2b$. In Alfano & Crisfield (2001) it is suggested that delamination can be treated as a two dimensional analysis in plane strain, assuming the width as neglected dimension.

In order to verify this assumption, in this subsection it is simulated a DCB in both plane stress and plane strain and the results compared. The elastic properties and the geometry are reported in table 6.4 and the experimental results taken from Chen *et al.* (1999). The distribution of nodes used is 8×774. Figure 6.25 shows that the plane strain state predict the peak load more accurately than the plane stress, while both can almost identically predict the post critical opening phase.

6.4.6.5 End Loaded Split (ELS)

Delamination of mode II (sliding mode) has been modeled as in figure 6.16 with a loading displacement applied at the free end of the bottom plate. For mode II loading, the interfaces are here supposed frictionless. Properties and geometry are summarized in table 6.4. Experimental results are taken from Chen *et al.* (1999), while the distribution of nodes used is 8×550.

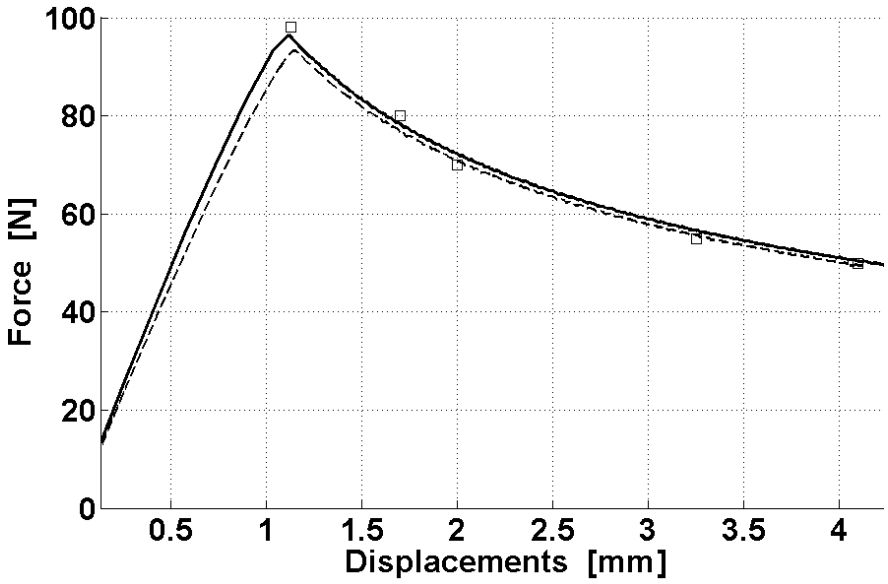


Figure 6.25: Force - Displacement curve for DCB test in Chen *et al.* (1999) : continuous line: numerical plane strain; dashed line: numerical plane stress; squared line: experimental

From figure 6.26 it can be observed that the model can capture the trend with a minor difference from the experimental test. In figures 6.27 and 6.28 it is pretty clear the presence of the damage and the stresses are distributed as expected. In fact, in the longitudinal stress plot (figure 6.27) , two zones can be readily identified: ahead the crack tip, the plate responds as an entire plate and the distribution is typical for a plate in bending; behind the crack tip, where there is separation, the longitudinal stress is representative of two distinct plates in bending.

In figure 6.28 instead it can be observed the stress concentration due to the presence of the delamination. The cohesive zone is shaped like a *bubble* and it can usefully track the progression of the delamination.

The critical opening displacement is around 15 mm, after that the delamination advances quite rapidly (figures 6.27 and 6.28). The delamination is complete (*split*) for a displacement of 21 mm, after that the two beams respond separately to the loads.

E_{11}	E_{22}	G_{12}	ν_{12}
130 GPa	8 GPa	5 GPa	0.27

(a) Elastic Properties

L	b	w
150 mm	1.55 mm	24 mm

(b) Geometry for DCB test

L	b	w
105 mm	1.525 mm	24 mm

(c) Geometry for ELS test

G_{Ic}	τ_n^{max}	a_0
257 J/m^2	20 MPa	22 mm

(d) Mode I interface properties

G_{IIc}	τ_t^{max}	a_0
856 J/m^2	48 MPa	60 mm

(e) Mode II interface properties

Table 6.4: Properties for material in Chen *et al.* (1999)

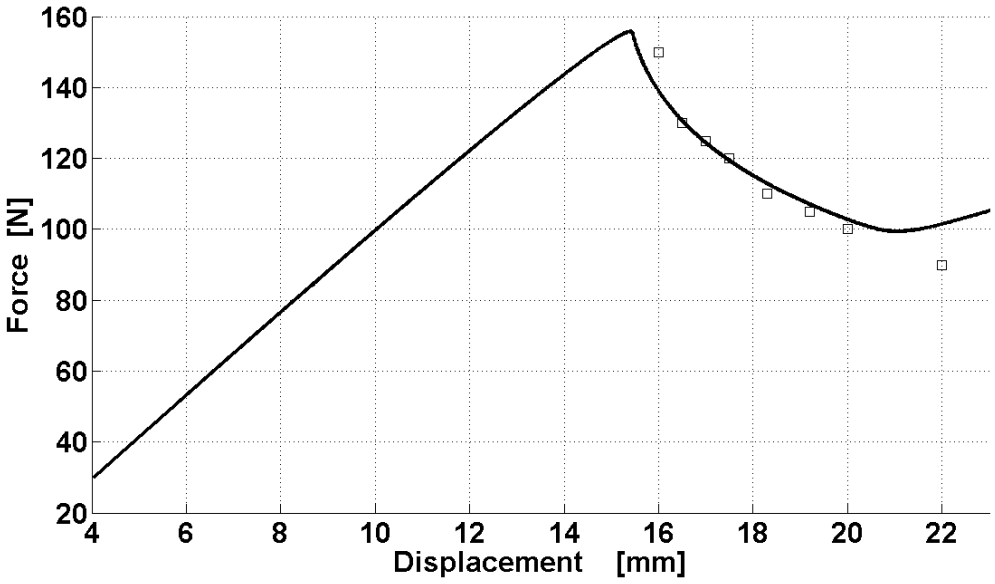


Figure 6.26: Force-Displacement curve for ELS test : continuous line: numerical ; squared line: experimental

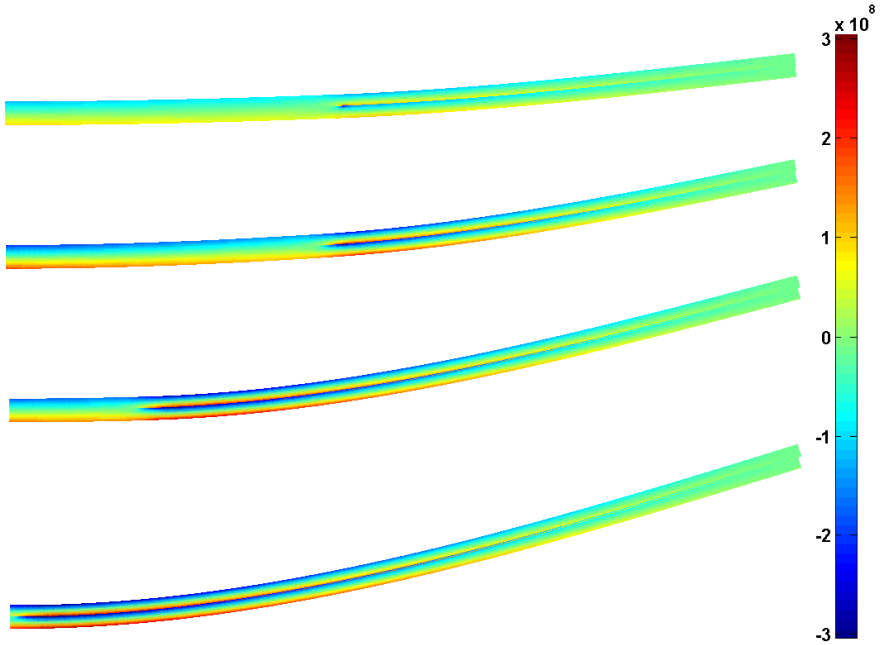


Figure 6.27: Delamination Stages for ELS test: longitudinal stress σ_x plot

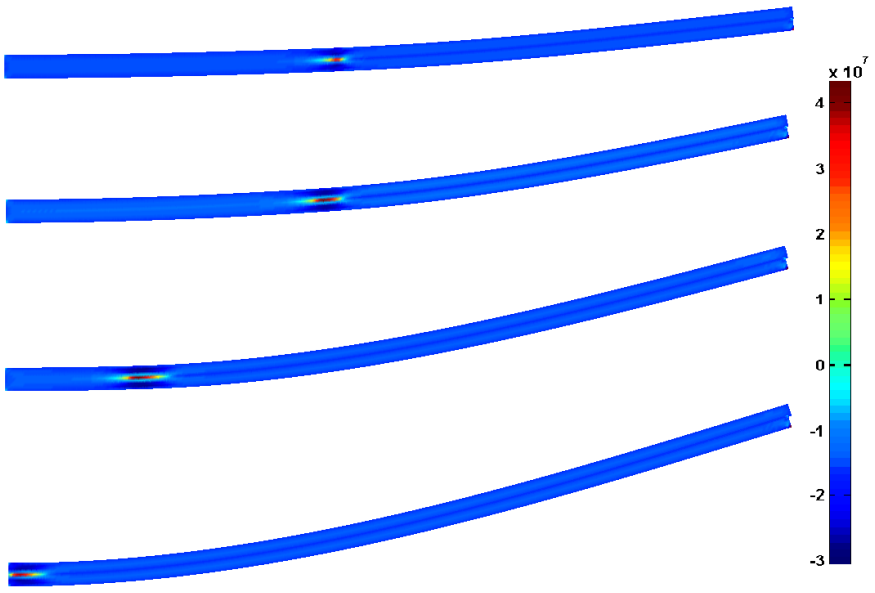


Figure 6.28: Delamination Stages for ELS test: longitudinal stress τ_{xy} plot

6.4.6.6 Mode II End Notched Flexure: AS4/PEEK

Another example of Mode II loading is an ENF test where a vertical force is applied in the middle of a simply supported beam with an initial delamination length (*three-point bending* as in figure 6.17) Experimental results are taken from Camanho & Dávila (2002) and geometry and properties (AS4/PEEK) 6.3 are the same of subsection 6.4.6.3. The distribution of nodes is 8×512. Once again a good agreement can be observed between the experimental and the numerical results. The critical opening displacement is perfectly predicted with a minor difference in the value of the peak force. Delamination phases can be observed from figures 6.30 and 6.31. From both pictures the effects of the delamination are pretty clear. Wherever the bonding is full, the plate responds to the load as a beam with a concentrated load in the middle, otherwise the disconnected parts behave independently. The beam is, as expected, in tension for the bottom and in compression for the top (6.30).This is true for both the areas ahead and behind the crack tip. The crack tip can be easily individuated also from the *bubble-like* shear stress concentration in figure 6.31. The last phase of picture 6.31 indicates full delamination and the shear stress is piecewise constant along the beam length, with a discontinuity, as expected, in the middle where the load is applied.

6.4.6.7 Mode II End Notched Flexure: T300/977-2

Final example for mode II is an ENF test for a graphite/epoxy material T300/977-2 (same specimen of DCB test in subsection 6.4.6.1 but with different width) taken from Dávila *et al.* (2001) whose elastic and interface properties are resumed in tables 6.1. The distribution of nodes used is 8×404. The figure 6.32 shows a comparison with FE decohesion elements which has been the most widely FE-based used technique for delamination modelling. As it can seen, results are very similar. Moreover the curve is much smoother than the previous case. This is mainly due to convergence issues as mentioned in the previous section 6.3.4.

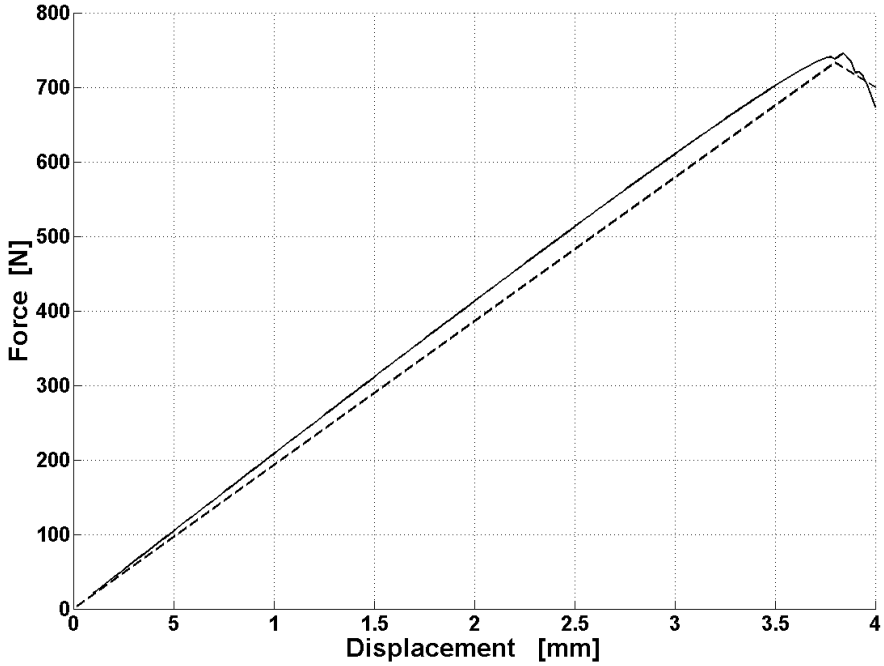


Figure 6.29: Force-Displacement curve for ENF test (AS4/PEEK) : continuous line: numerical ; dashed line: experimental

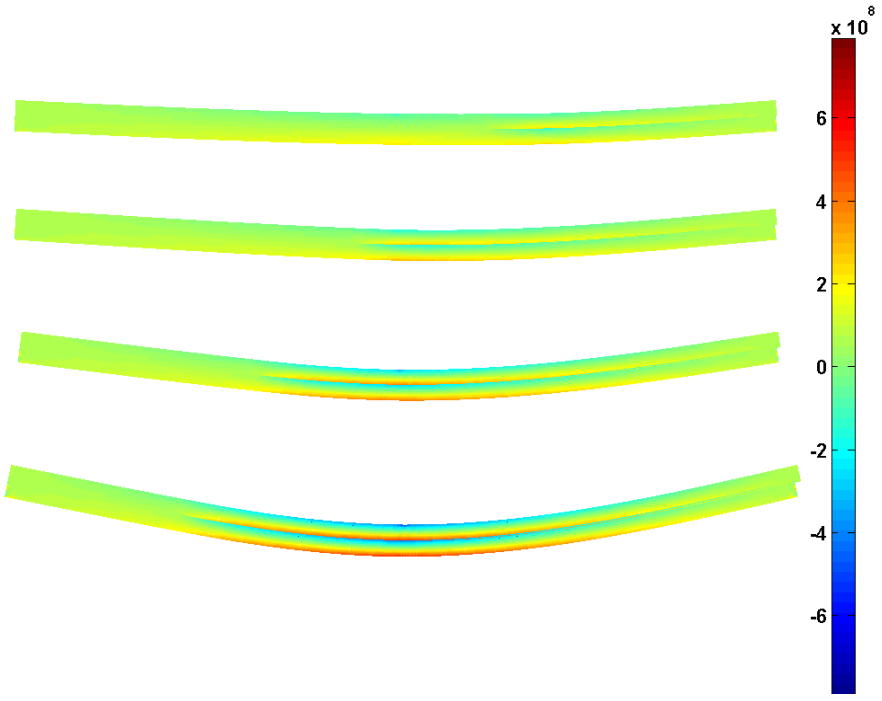


Figure 6.30: Delamination Stages for ENF test: longitudinal stress σ_x plot

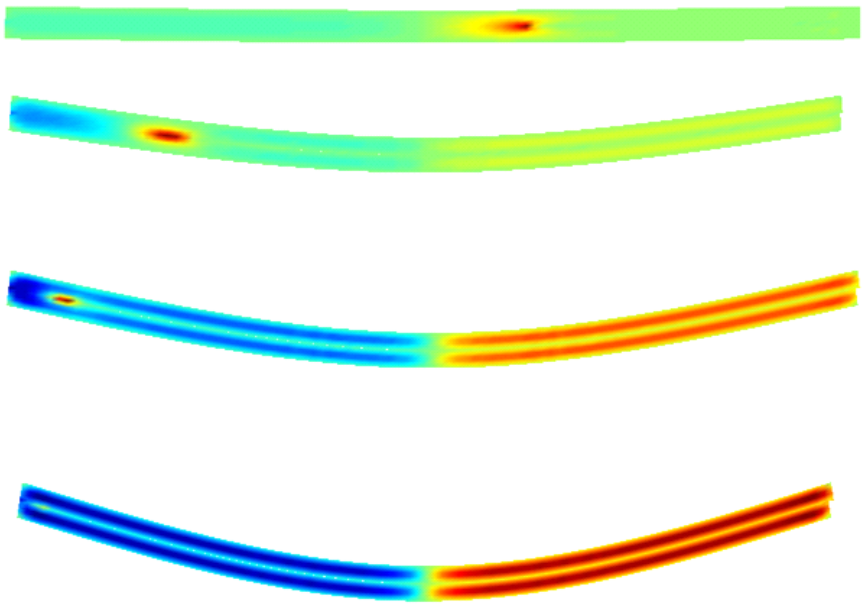


Figure 6.31: Delamination Stages for ENF test: shear stress τ_{xy} plot

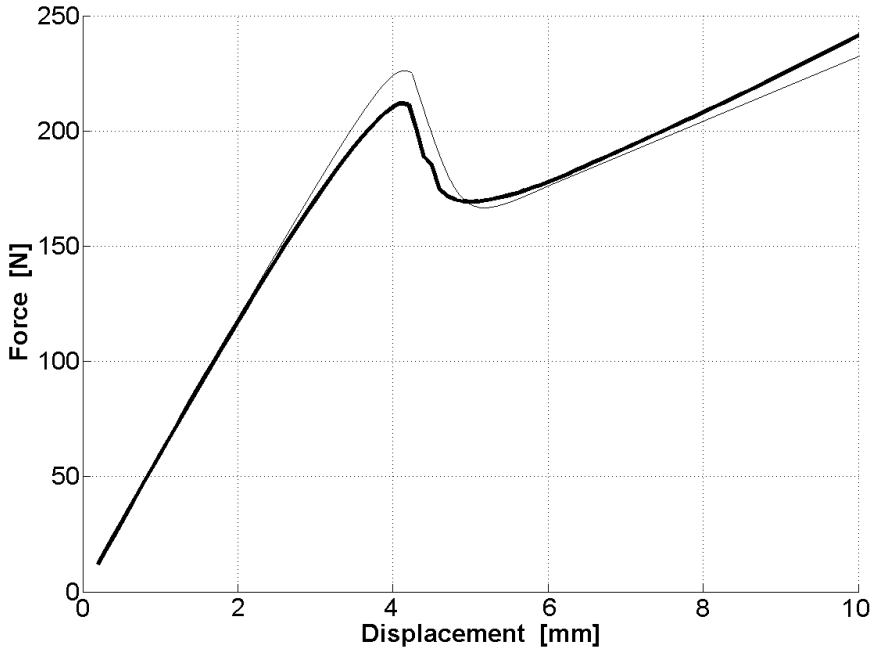


Figure 6.32: Force-Displacement curve for ENF test (T300/977-2) : continuous thick line: RKPM ; continuous thin line: FE decohesion

6.4.6.8 Central delamination with no pre-crack

The next case is a loading mode I delamination with no initial pre-crack. The specimen is cantilevered at the ends and transversely loaded, as in figure 6.19. The specimen is the same for Mode I in table 6.4 except that $a_0 = 0$. The cohesive approach allows the initiation and subsequent propagation of the crack, as it can be seen from the first phases of delamination in figures 6.33, 6.34 and 6.35. The complete equilibrium path is shown in figure 6.36. The first part of the curve (in figure 6.37) is an *elastic* phase with high stiffness and very small displacements (from 0 to 500 N), since the material is supposed undamaged and a relatively high force is necessary to overcome the cohesive tractions at the interface. After 500 N the response is *plastic-like*, until the delamination starts for a peak load of almost 1500 N and an opening displacement of less than 0.05 mm. Subsequently the delamination propagates symmetrically respect to the middle, with patterns similar to the DCB tests examined previously, until complete delamination, when the two beams split (around 2.25 mm).

6.4.6.9 Double Cantilevered Beam with multiple cavities

As anticipated in subsection 6.3.4, there are some numerical issued due to a characteristic length related to the cohesive zone, namely the cohesive length. This issue affects the nodes distribution, since it should be efficiently fine to capture the concentration of the stress, otherwise the under-sampling can alter the equilibrium path. The consequences are normally two-fold: the arrest of the numerical continuation and the appearance of spurious bumps in the equilibrium path. In figure 6.38 is shown the path for the DCB case in 6.4.6.1 with a coarser *mesh*. The false instabilities appear after the initiation of the delamination, although the peak load is predicted very accurately. Therefore, if the main purpose of using the code is to estimate the peak load and the critical opening length, a coarser mesh, which is computationally inexpensive, would seem sufficient. Nevertheless, the following examples will show that this approach fails due to the presence of instabilities too close to the initial crack and the peak load might not be predicted accurately.

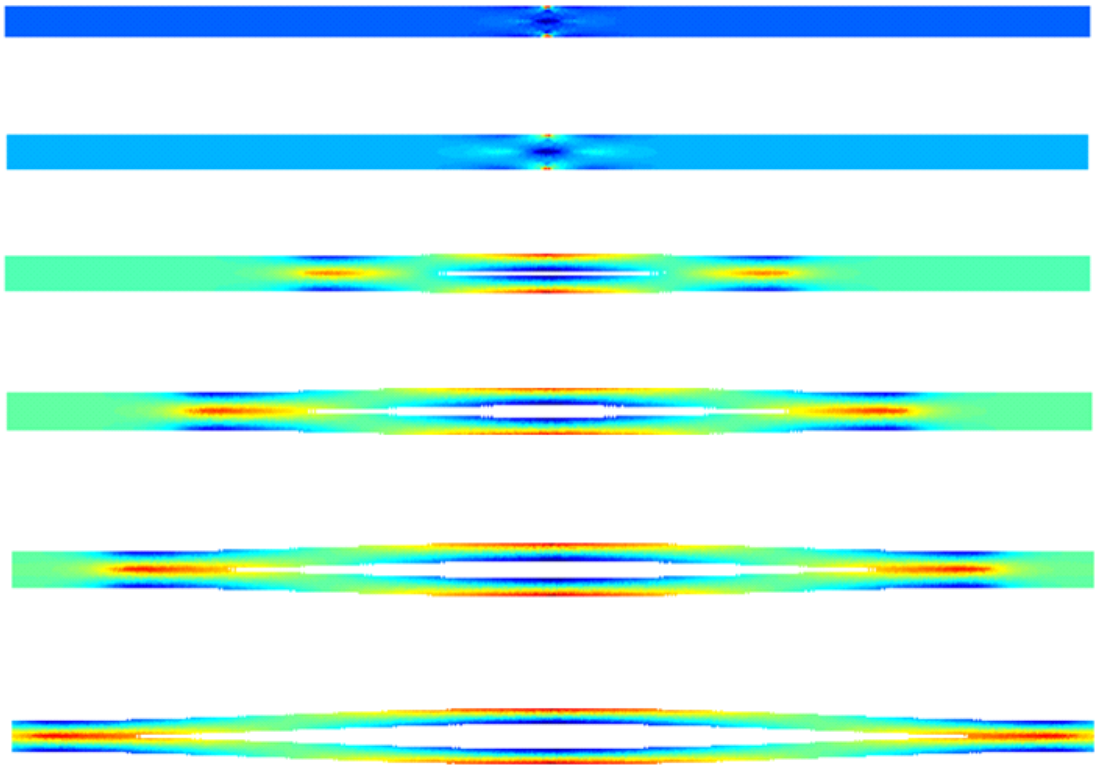


Figure 6.33: Delamination stages for the case of central delamination: longitudinal stress σ_x plot

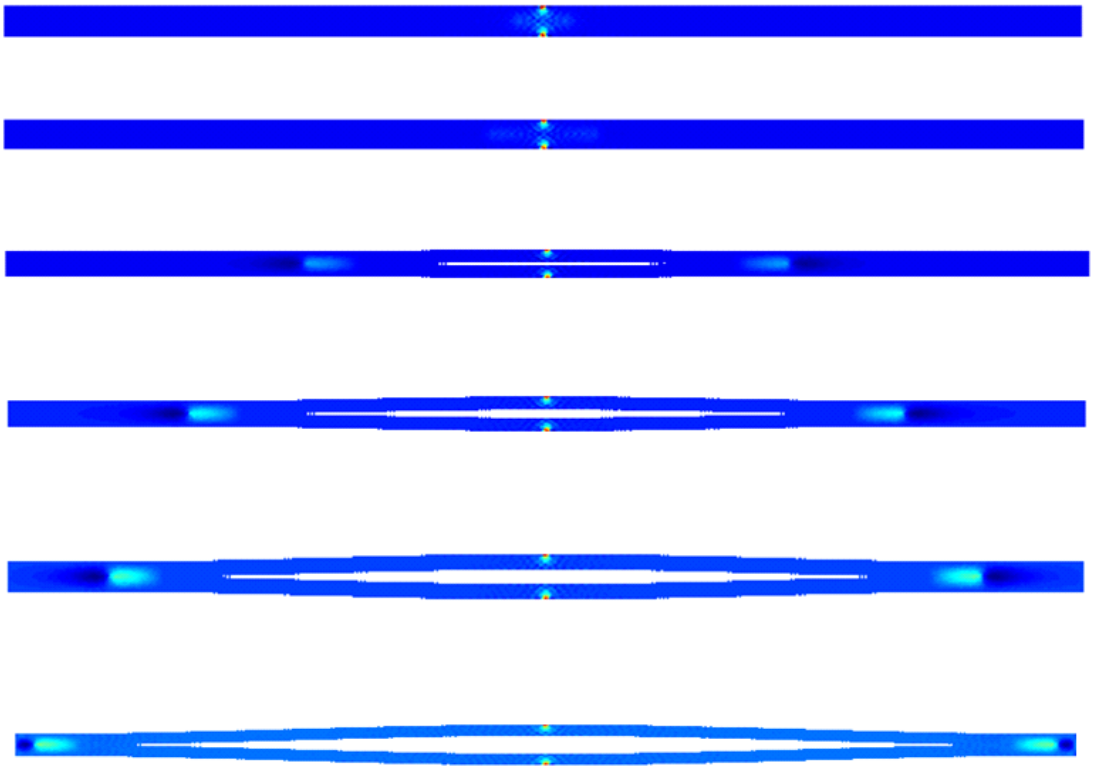


Figure 6.34: Delamination stages for the case of central delamination: longitudinal stress σ_y plot

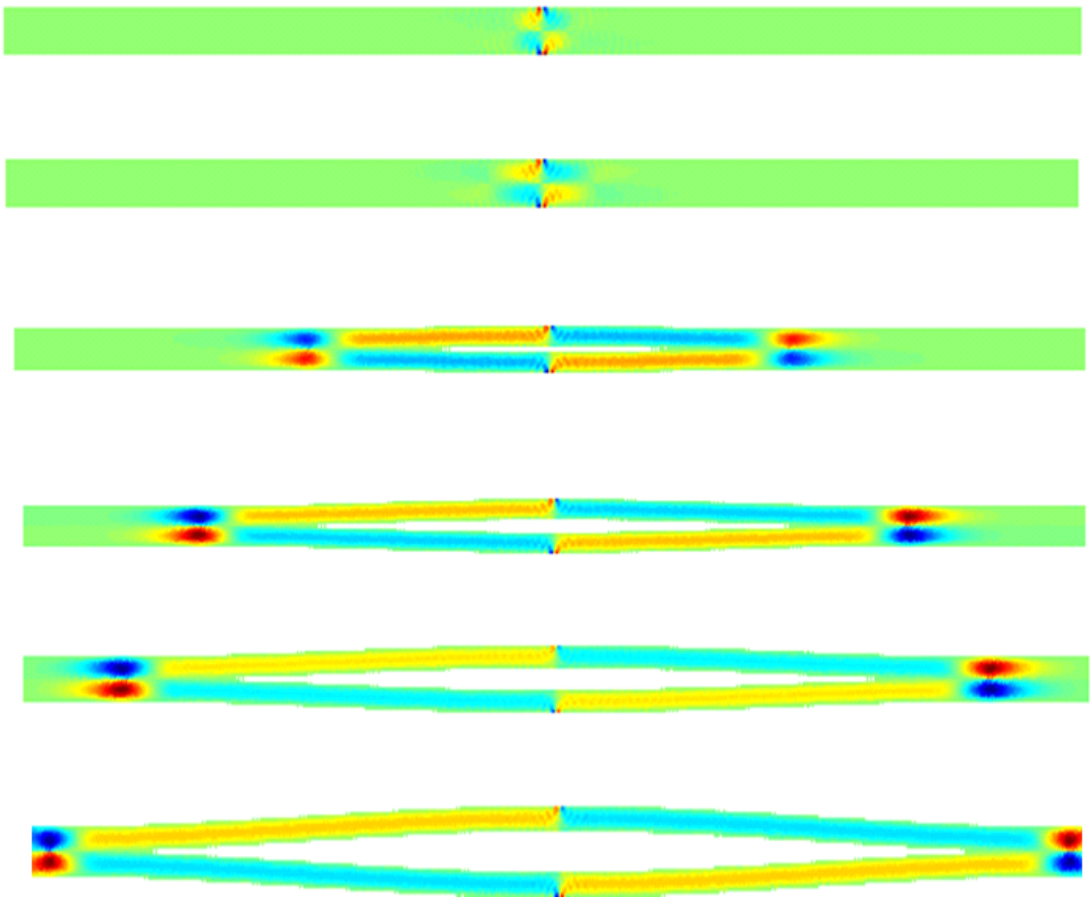


Figure 6.35: Delamination stages for the case of central delamination: longitudinal stress τ_{xy} plot

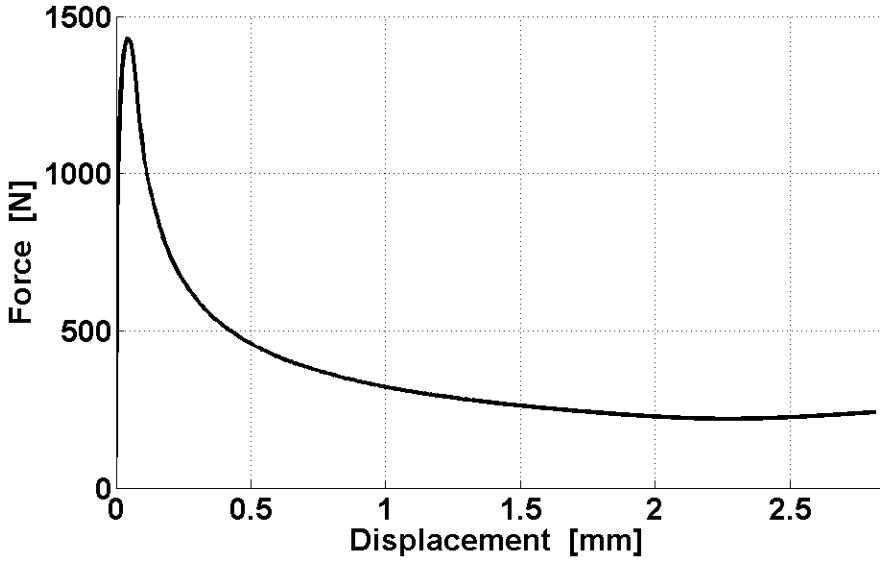


Figure 6.36: Force-Displacement curve for central delamination

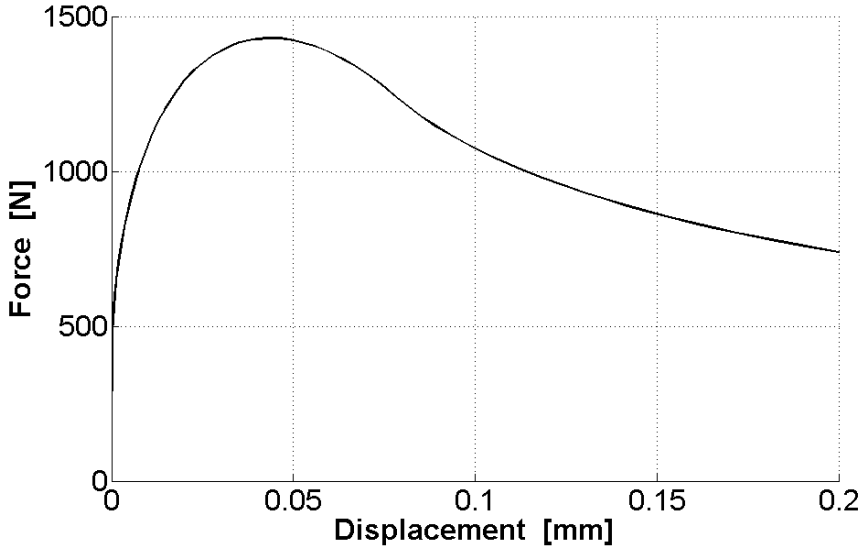


Figure 6.37: Force-Displacement curve for central delamination (zoom on the initial phase)

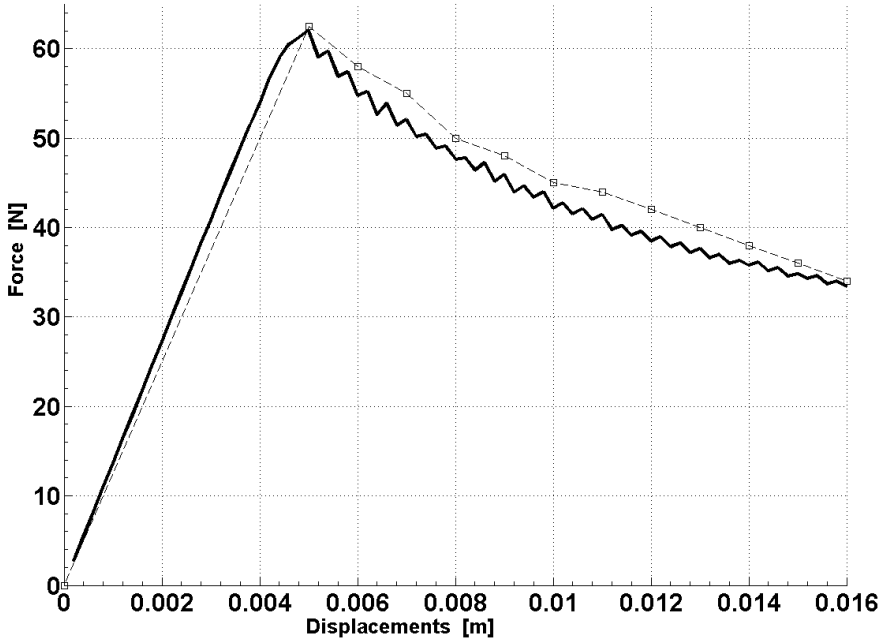


Figure 6.38: Force-Displacement curve for DCB test T300 with coarser mesh 5×378

In Samimi *et al.* (2009) is reported that these local bumps are caused by sudden release of strain energy. This release is due to simultaneous failure of areas of the interface wider than the cohesive length. The areas are more precisely elements, if too few it may happen more elements than necessary may improperly satisfy the separation criterion and therefore cause de-bonding. To test this statement, the false instabilities were simulated with *failures* in the contact, .i.e. the penalty stiffnesses were set to zero locally (as in figure 6.39) and the position and the number have been varied. The case tested is a DCB for an isotropic material in plane stress. The elastic properties of the material and the geometry are resumed in table 6.5 and the distribution of nodes is 11×1100.

The first example is represented in figure 6.39 where a first pre-crack is present and internal failures have been introduced periodically. The length of the failed zone is the same of the pre-crack (1 mm) and their separation is equal to the length of the pre-crack.

In figure 6.42 the presence of multiple failures is manifested by wide bumps in the curve that occur when the delamination reaches a failed segment. Around a

E	ν
100 MPa	0.3

(a) Elastic Properties

G_{Ic}	τ_m	a_0
100 J/m^2	1 MPa	1 mm

(b) Interface properties

L	b	w
10 mm	1 mm	1 mm

(c) Geometry

Table 6.5: Properties for DCB test with cavities

displacement of 3 mm, the continuation interrupted as convergence of the model was not further achieved. By interruption it is meant that the continuation code, being a path following technique, tries to reduce the step-length on the equilibrium path. If convergence is not achieved, the step-length (a value of 10^{-5} m is usually a good choice) is reduced further until a convergent state is found. Usually a very small length is set as an inferior bound (for example 10^{-15} m), below which if convergence is not found, then it is convenient to interrupt the continuation. In fact, it has been experienced by the author that setting the inferior bound even lower, does not help in overcoming the obstacle.

Figure 6.43 shows the shear stress at the last convergent state. It can be seen that the presence of the failures alters the stress distribution with respect to the shear stress distribution showed in figure 6.23b. In fact, the next failed segment introduces a sort of *border* effect that increases the stress concentration at its ends. The more acute concentration of stress further reduces the cohesive length, therefore more nodes (or element) are necessary to capture the concentrated field and proceed with the numerical continuation.

The peak loads and the critical opening displacements are the same predicted from the *undamaged* model. This is not the case for the following example (figure 6.41), when only one internal failure of reduced length 0.25 mm is introduced at a distance 2 mm from the free end. In fact, even though the curve is the same, the *damaged* model has a lower peak force (figure 6.44) and an earlier critical opening displacement. This is due to the fact that the internal failure is *close* to the pre-crack. In fact, if the cavity is inserted far from the crack-tip (figure 6.41,

in the middle of the plate), the equilibrium path 6.45 is identical to the one with no internal failures, except, of course, when the delamination reaches the failed zone.

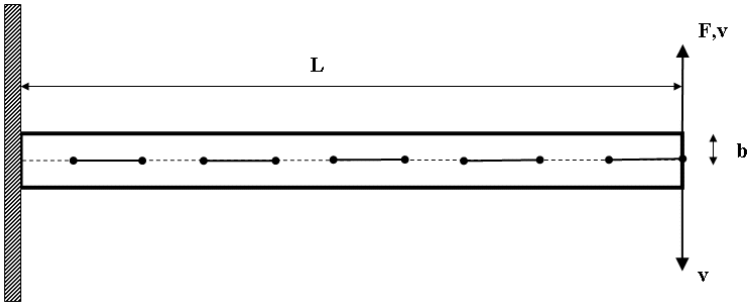


Figure 6.39: DCB with multiple cavities: continuous line: failed interface; dotted line: undamaged interface

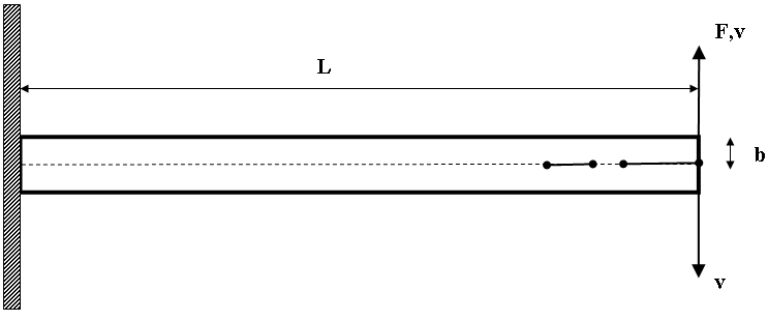


Figure 6.40: DCB with one cavity close to the crack-tip

In conclusion, to sum up:

- the presence of sudden localized failures, as in insufficient sampling of the interface, can lead to local bumps in the equilibrium path;
- the continuation might *stuck* on these false instabilities due to excessive stress concentration;
- if a pre-crack exists, if the failure is located close to the initial pre-crack, the two cracks interact. In this case the peak load and the critical opening displacement might not be predicted accurately: therefore the use of a coarse mesh, even if less computationally expensive, could be inaccurate.

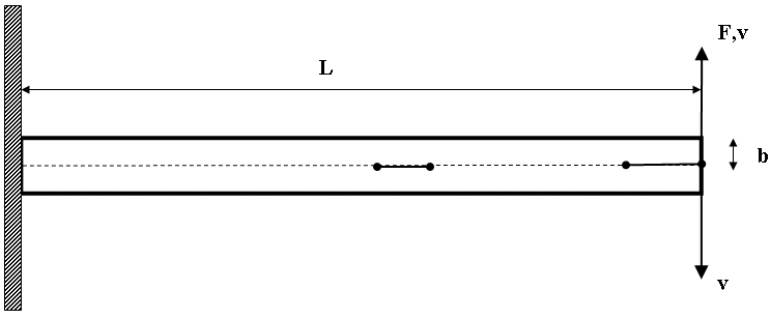


Figure 6.41: DCB with one internal cavity

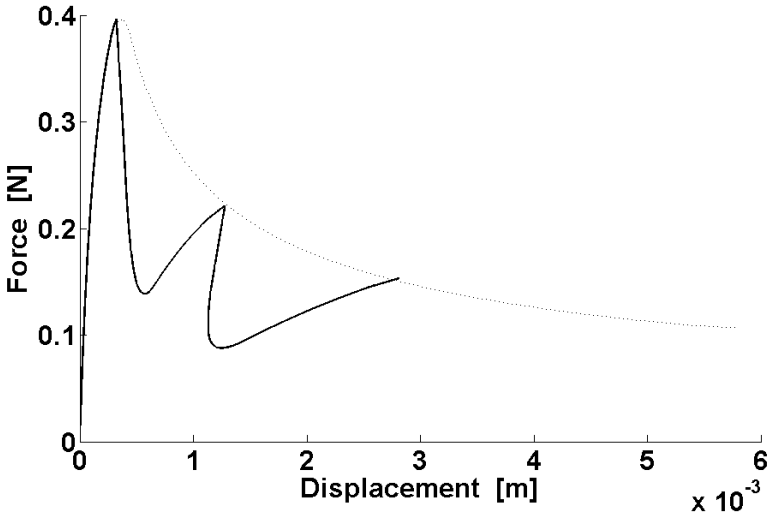


Figure 6.42: Force-Displacement curve for DCB with cavities: continuous line: pre-crack and multiple failures ; dotted line: pre-crack and no failures

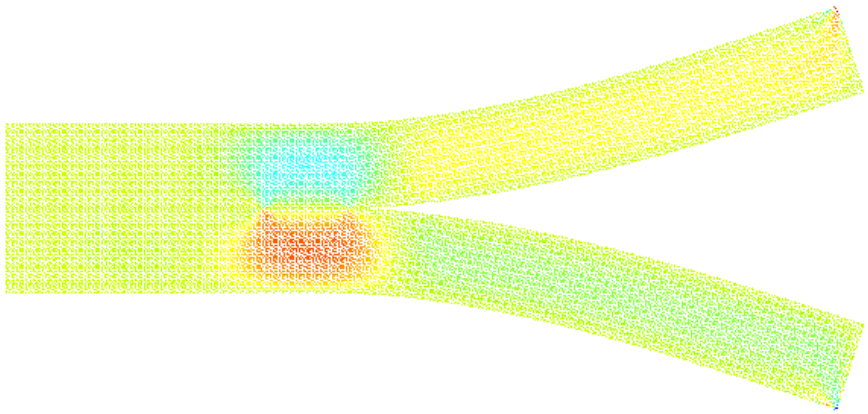


Figure 6.43: Shear Stress τ_{xy} plot for DCB with pre-crack and multiple failures

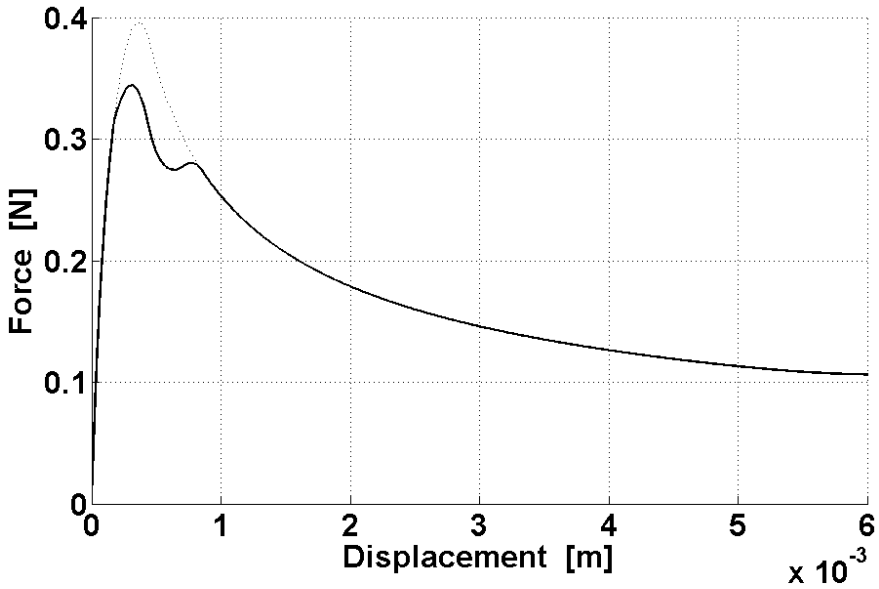


Figure 6.44: Force-Displacement curve for DCB with one cavity as in fig. 6.40: continuous line: pre-crack and one failure ; dotted line: pre-crack and no failures

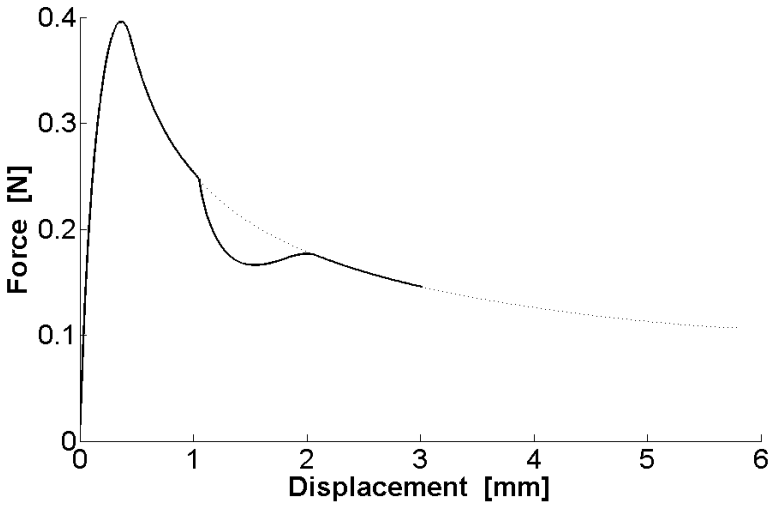


Figure 6.45: Force-Displacement curve for DCB with one cavity far from the crack-tip as in fig. 6.41: continuous line: pre-crack and one failure ; dotted line: pre-crack and no failures

6.4.7 Comparison with the Penalty Approach

In this subsection a comparison between the two approach is presented. It will be shown that the cohesive segments method leads to almost identical results to the penalty approach. The advantage of cohesive segments, though, is that crack modelling is not restricted to the boundaries of sub-domains, i.e. when the crack path is not known *a priori*. For the penalty approach, 8 nodes in the thickness for each sub-plate were used. The number of nodes in the length is chosen accordingly to equation (6.131). Numerical Gaussian quadrature of order 3 was performed over triangular elements and over the cohesive segments, where *ad-hoc* quadrature cells are constructed in the class `Cohesive` (section 6.4.9).

Figures 6.46, 6.47 and 6.48 show a comparison between the penalty approach and the cohesive segments method. As it can be seen, results are indistinguishable, apart from figure 6.47 where there is very little difference in the prediction of the force in the post-critical phase. Nonetheless both methods agree quite well with the experimental findings.

The penalty method is based on a sub-division of the domain, while the cohesive segments superimposed a discontinuity. The fact that both methods give

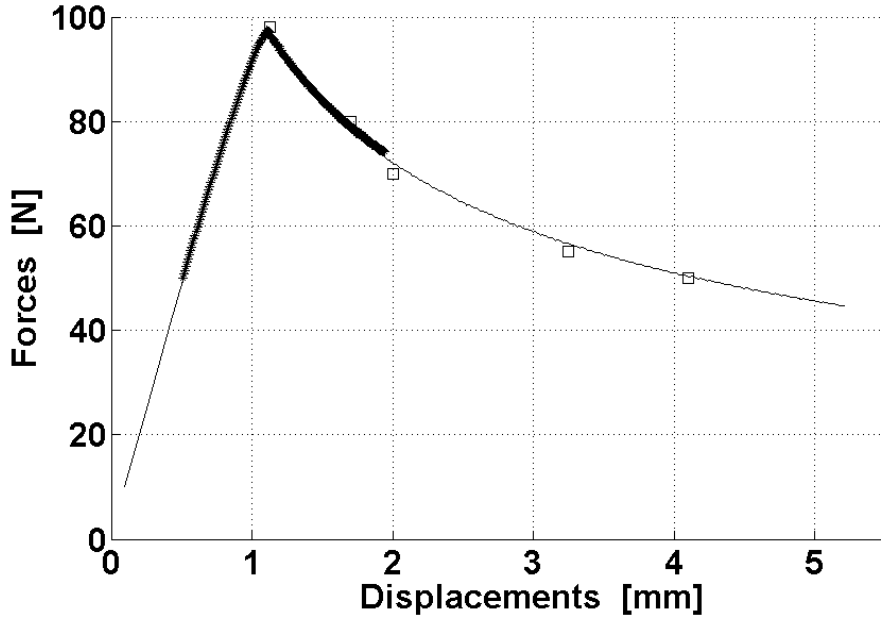


Figure 6.46: Cohesive Segments DCB : Force - Displacement curve; continuous line: cohesive segments; squares: experimental; crossed line: penalty approach

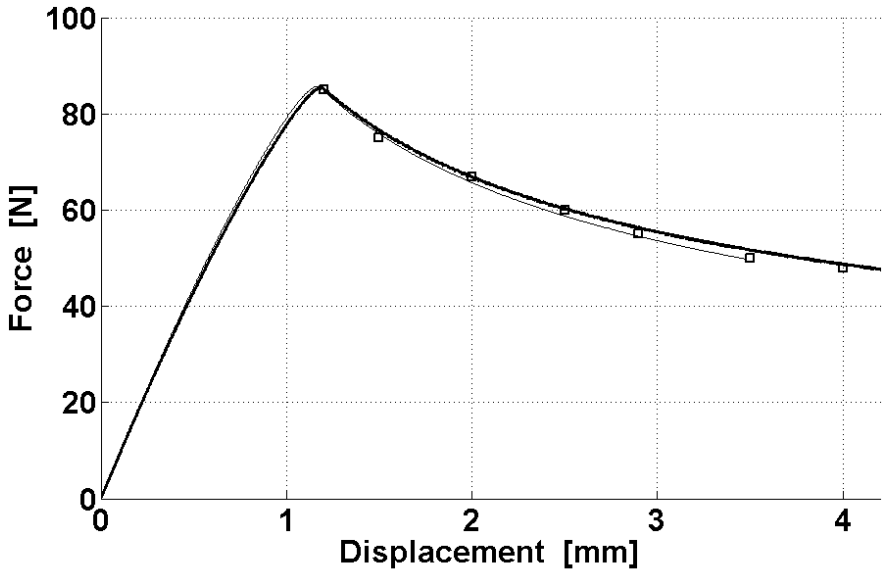


Figure 6.47: Cohesive Segments DCB XAS-913C: Force - Displacement curve; continuous thin line: penalty approach; squares: experimental; continuous thick line: cohesive segments

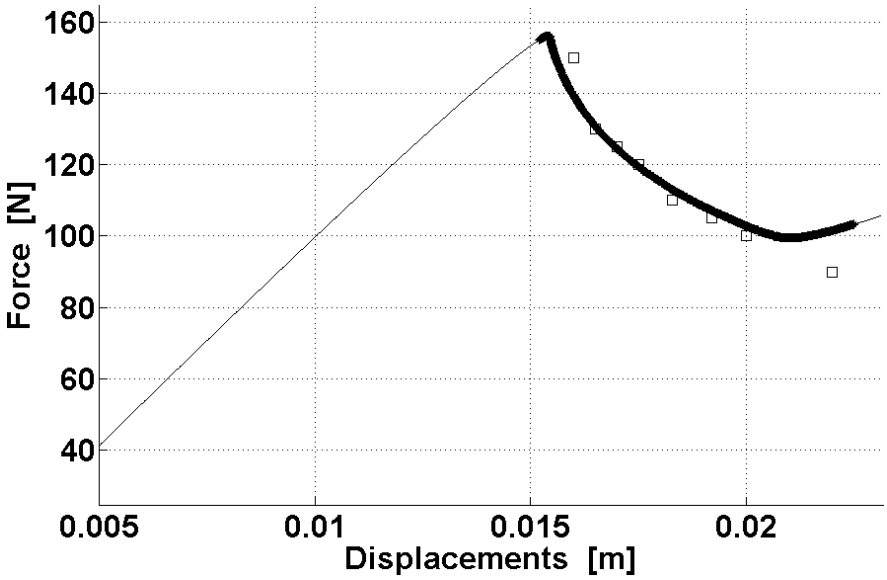


Figure 6.48: Cohesive Segments ELS: Force - Displacement curve; continuous line: cohesive segments; squares: experimental; crossed line: penalty approach

the same results is an indication that the cohesive segments method is very effective in reproducing the strong discontinuity caused by the delamination, which in the penalty method is introduced natively by construction of two different sub-domains.

6.4.8 Convergence issues

In this section some issues encountered during the numerical continuation of the non-linear equations will be described.

6.4.8.1 Effects of the inter-laminar strength τ^{max}

In figure 6.49 it is shown a comparison for test in section 6.4.6.4. The parameters varied are the inter-laminar strength τ_n^{max}, the state of plane stress or plane strain and the nodes distribution. In all cases, the propagation of the delamination is represented with quite good agreement with the experimental results. Nevertheless, the prediction of the peak force and the critical opening can fail due to the above mentioned parameters. In fact, for same discretization and value of τ_n^{max},

better results are obtained if the plane strain is assumed, as shown also in section 6.4.6.4. From a *cohesive-length* point of view, the use of the plane strain lengthens the cohesive length, since it is proportional to the elastic modulus. Indeed, for equation (4.99), the elastic moduli are augmented when plane strain is used.

The reduction of τ_m is normally used to stretch the cohesive zone in coarse mesh. Nonetheless, figure 6.49 shows that this *trick* might lead to inaccurate prediction of the peak load. In fact, for same stress state and discretization, the use of a higher inter-laminar strength improves the prediction to almost the experimental one. Finally, the thick line in figure 6.49 is the result slightly higher inter-facial strength. A finer mesh better samples the cohesive process and ultimately lead to an accurate prediction of the peak load.

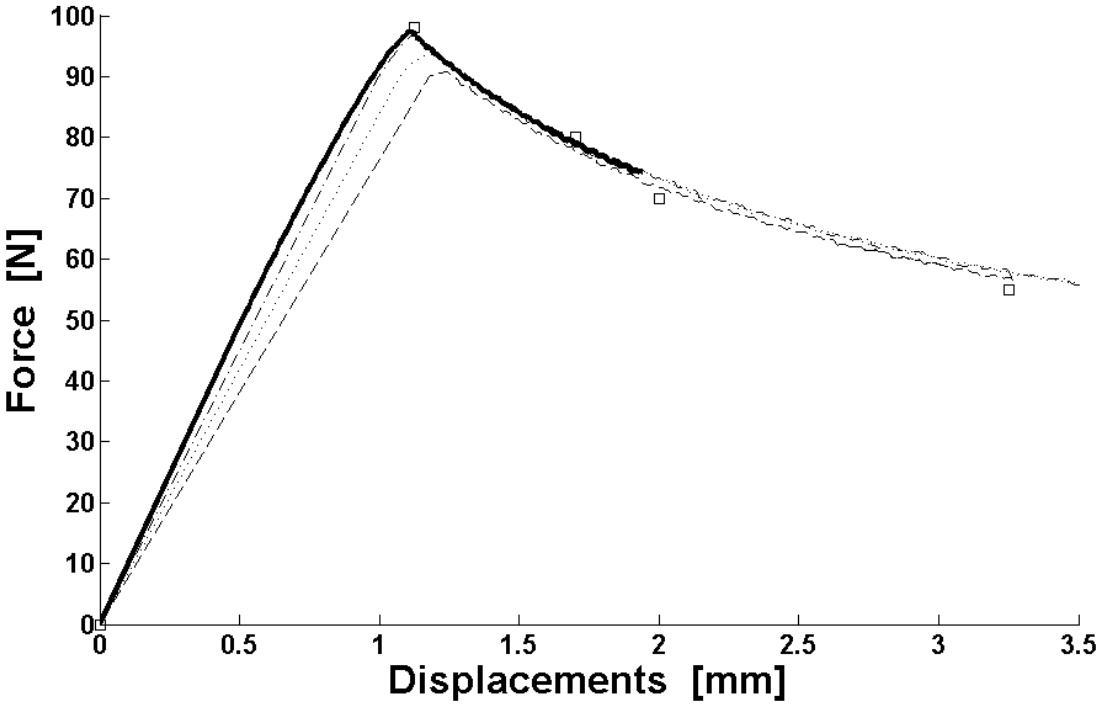


Figure 6.49: Convergence issues: squares: experimental data; continuous line: 8×774 plane strain, $\tau_n^{max} = 17$ MPa; dash-dot line: 6×580 plane strain, $\tau_n^{max} = 15$ MPa; dotted line: 6×580 plane strain, $\tau_n^{max} = 10$ MPa; dashed line: 6×580 plane stress, $\tau_n^{max} = 10$ MPa

6.4.8.2 Effects of the critical strain energy release rate G_c

The reduction of the inter-laminar strength can sometimes be beneficial, although, as seen in the previous subsection, it should be used *carefully*, as an excessive reduction can lead to incorrect results.

Instead, the strain energy release rate G_c should not be changed arbitrarily. In fact, figure 6.50 shows the equilibrium path for the test in section 6.4.6.1, where a $G_{Ic} = 352\ J/m^2$ has been used, as reported in Turon (2007). If a reduced value it is used, the equilibrium path is completely different and incorrect. The value used in curve 6.50 is $G_{Ic} = 268\ J/m^2$.

Indeed, since the cohesive zone model is primarily an *energy-based* model, reducing the value of G_c means reducing the area under the triangle in figure 6.6. This means that the critical failure will happen to a lower peak force and for a lower critical opening length, as in figure 6.50.

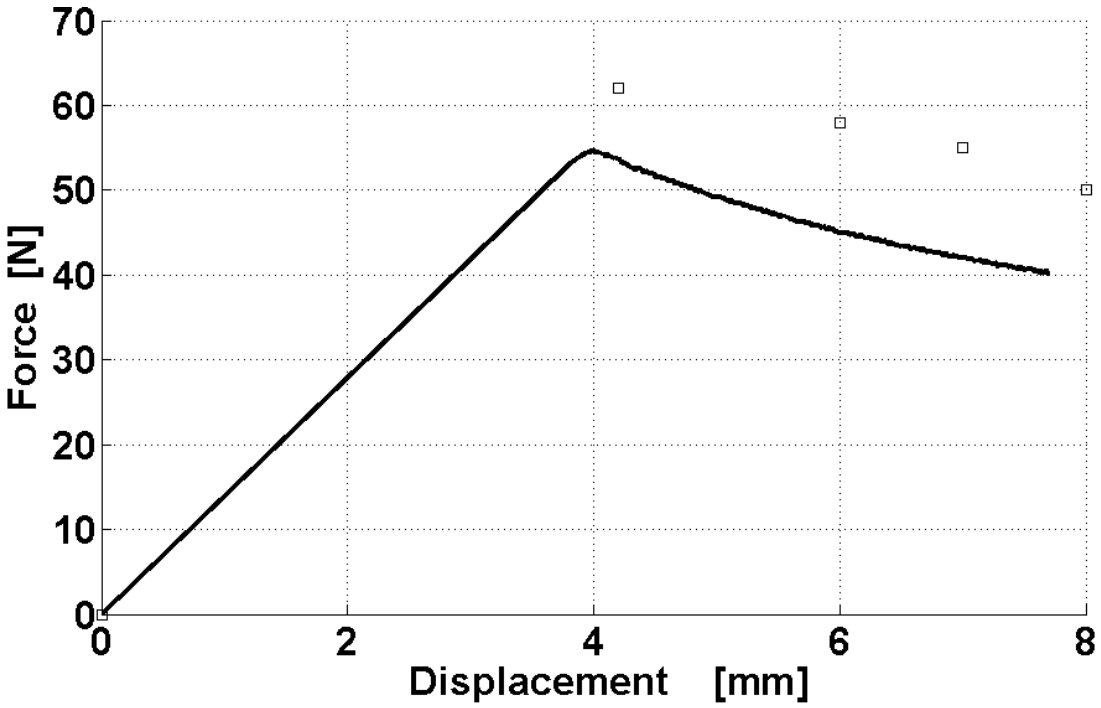


Figure 6.50: Convergence issues for different G_c: continuous line: numerical; squares: experimental

6.4.8.3 Effects of a coarse mesh

The effects of a coarse mesh have been detailed in subsection 6.4.6.9, usually accompanied by the *arrest* of the continuation. Another curious effect is the *bouncing back* of the continuation, as in figure 6.51. The test in the example is the ELS of subsection 6.4.6.5. When the critical force is reached (unstable point), instead of starting the delamination, it can happen that the continuation goes back to the previous convergent path. One possible explanation is that the continuation code is used in combination with a *line-search* algorithm. The line search looks for a reduction of the residual error. If this reduction cannot be found after the critical point (i.e. for the difficulties due to a coarse mesh), the only way to reduce the error and find the equilibrium is then to *go back* to the previous convergent path, which is for sure an equilibrium path.

The result is then that no new equilibrium states are found but the continuation rebounds on old equilibrium states.

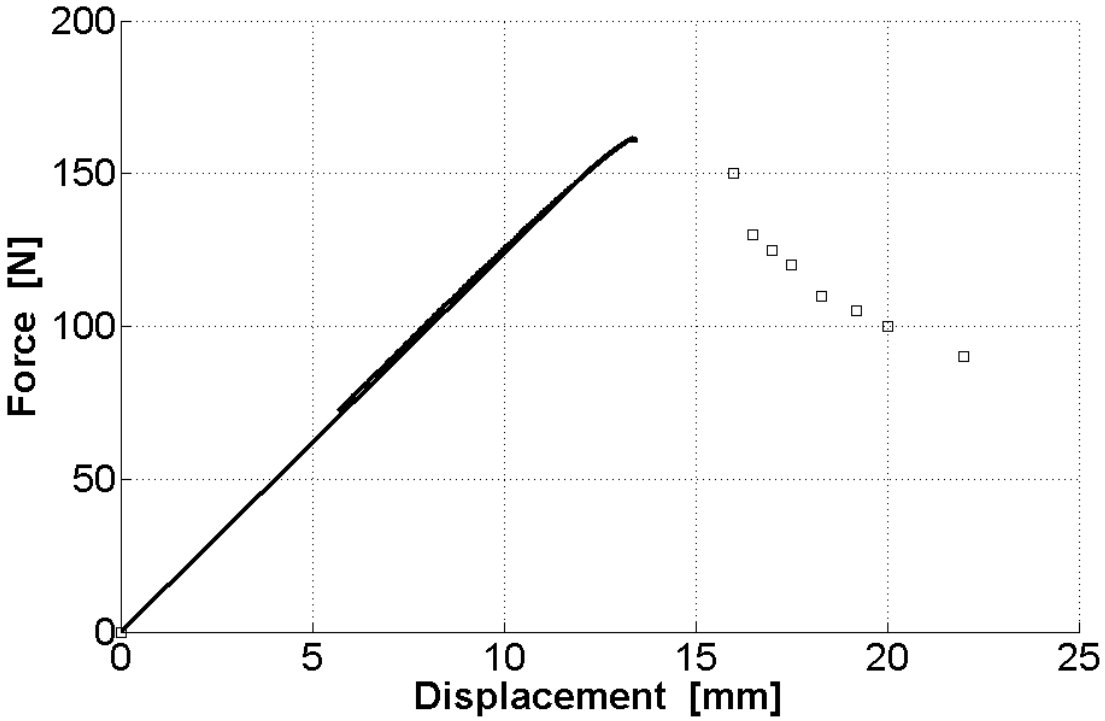


Figure 6.51: Bounce back of the continuation

6.4.9 The class `Cohesive`

6.4.9.1 The properties

Many properties are the same defined for the class `CohesiveContact`.

- `d0t`, `dFt`, `d0n`, `dFn`, `GIc`, `GIIc`, `smax_I`, `smax_II` are the parameters of the cohesive zone model, as described in section 6.2;
- `dist` a structure that contains the functions in equation (6.126),(6.127),(6.128), (6.129) and (6.130);
- `H` a function handle containing the enrichment function (6.62);
- `nseg`: the number of cohesive segments;
- `Ind` a cell array of size `nseg`, where each cell contains the indices of the enriched nodes for each segment, selected according to equation (6.120);
- `P0` an array size `nseg` $\times$ 2, containing the coordinates of the initial point of the segment, as in figure 6.10;
- `theta` an array size `nseg` containing the inclination of each segment, as in figure 6.10;
- `L`: an array size `nseg` containing the length of each segment, as in figure 6.10;
- `tx` and `ty`: cell array of size `nseg`, where each cell contains the tractions at the interface as defined in equation (6.21);
- `vx` and `vy`: cell array of size `nseg`, where each cell contains the displacement jumps at the interface as in equation (6.21);
- `hn`, `ht` cell array of size `nseg`, where each cell contains an array defined on the Gaussian points of the segment, with values 0 if there is a pre-crack, 1 otherwise;
- `KTt` and `KTn` cell array of the derivatives of the penalty factors;

- `Krc` cell array of size `nseg` $\times$ 1 containing the matrices in equation (6.88) ;
- `Kcc` cell array of size `nseg` $\times$ `nseg` containing the matrices in equation (6.89);
- `Vrc` cell array of size `nseg` $\times$ 1 containing the matrices in equation (6.93);
- `Vcc` cell array of size `nseg` $\times$ `nseg` containing the matrices in equation (6.95);
- `KT` cell array of size `nseg` $\times$ `nseg` containing the matrices in equation (6.114);
- `Fc` cell array of size `nseg` $\times$ 1 containing the vector in equation (6.110);
- `Segments` is a structure composed of five fields:

    1. `Omega`: cell array of size `nseg` $\times$ 1, where each cell contains a RKPM object, defined on the Gaussian points of Ω, with RKPM nodes defined as the selected enriched nodes contained in `Ind`, as in equation (6.66) and (6.67).

       In this case, since the `PSI`, `dPSIcsi`, `dPSIeta`, `ig` and `js` vectors are already available; they can be taken from the existing object `Plate`. Therefore, the methods `GetFromPlate` and `ShapeAndDerWithPsiPU` are used.

       Subsequently, the enrichment function is imposed using the internal function `AddH` that multiplies the shape functions and their derivatives by the *Heaviside* function in equation (6.62).

       These *enriched* shape functions are necessary to compute the matrices `Krc` and `Kcc`;

    2. `Line`: cell array of size `nseg` $\times$ `nseg` where the cell `Line{i,j}` contains a `RKPM` object that calculates the shape functions on the Gaussian points of the *j-th* segment but using the set of enriched nodes for the *i-th* segment. In this case, the method `ShapePU` is used. If no Gaussian points of the *j-th* segment are found within the supports of the nodes for the *i-th* segment, the cell is left empty and no shape functions are calculated;

3. `B`: cell array of size equal to the number of edges, where each cell contains a nested cell array of size `nseg`. Each nested cell contains a `RKPM` object. For example, `B{k}.RKPM{i}` computes the shape functions on the *i-th* edge of $\partial\Omega$, but using the set of enriched nodes for the *i-th* segment. If no Gaussian points of the *i-th* edge are found within the supports of the nodes for the *i-th* segment, the cell is left empty and no shape functions are calculated;

   This properties is used to calculate the force vectors in equations (6.93), (6.95), (6.101)

4. `PointForces` and `PointConstraints` same as the previous field, but for concentrated loads and point constraints.

6.4.9.2 The methods

The methods defined on the class are:

- `Cohesive` the constructor of the class. This method selects the node to enrich as in section 6.4.5 and sets up the integration points for the segments using the mapping in equations (5.7) and (5.8);
- `DisplJump` calculates the displacement jump as in equation (6.68) for each segment;
- `Assembly` calculates the stiffness matrices `Krc` and `Kcc`, the constraint matrices (6.93), (6.95) and the force vectors (6.101);
- `PenaltyStiffness` calculates the penalty factors according to the displacement jumps, as in equation (6.21);
- `AssemblyCohesive` calculates the cohesive force vector (6.110) and the tangent stiffness matrix (6.114);
- `PlotSegments` and `PlotEnrichedNodes` are graphic methods that plot the Gaussian points of the cohesive segments and the enriched nodes, as in figures 6.13 and 6.11.

Chapter 7

Conclusions and future works

The research has been focused mainly on exploring the possibilities offered by the meshfree techniques. The topic of major interest has been the modelling of failure characterized by strong discontinuities in composite materials.

Primarily, it has been shown that the meshfree methods can successfully simulate delaminations for different loading modes, without using finite elements and/or re-meshing. Quite good agreement has been found with experimental evidences taken from the literature. Most importantly, it has been shown that discontinuities can be introduced in the model without the need of modelling separate domains or the usage of interface elements, de-cohesion elements or other finite element-based technique. Compared to these methods, it has been demonstrated that delamination can be effectively modelled using meshfree with enrichments, or *extended* meshfree method.

The success in this sense it is also due to the combination of the meshfree capabilities with a constitutive relationship known as cohesive zone model. The cohesive zone model establishes a traction-displacement relationship between the cohesive traction at the interface of a crack with its opening displacement. The main purpose of cohesive zone models is to give a more physical explanation of the failure. This is different from classical linear fracture mechanics, which is not suitable to composites, as it involves non-physical crack tip stress singularities. Delamination is essentially a cohesive process and the possibility in the cohesive models to introduce *debonding* is a key factor in modelling delamination. The study of delamination was possible through the development of *prototype* meshfree

software.

The codes have been modified and improved from the very first *naive* one-dimensional implementation, which was honestly quite slow. Since then, the primary drive for continuous search of improvements has been the computational run-time. The results of this search have been proposed in this work, showing that with appropriate programming and mathematical manipulation, it is possible to enhance the performance of the meshfree methods, notoriously known as *good-but-slow* methods.

Another drive was the idea of designing a code that would have been similar to a commercial one. Therefore the efforts were directed to build a more sophisticated code. At the present time, the MM codes developed so far can handle dynamic and static problems in 2D and 3D even with complex geometries. The produced code is far from perfection, but hopefully would help the development of even more sophisticated (maybe industry-oriented) codes.

Particularly helpful in this sense was the *object-oriented* programming, that it is a flexible and intuitive philosophy of programming. The object-oriented programming allows the creation of *virtual models* of the object under study, and acting on it as it was a real object. Moreover, an object-oriented code facilitates abstraction and opens to future improvements and modifications, by creating new objects over existing ones, or brand new objects, and let them interact. For example, the objects for the cohesive crack were created upon the existing linear elastic object.

Fracture mechanics applications are so far limited to two-dimensions, due also to the fact that it is still an open problem in the scientific community and there is little knowledge on three-dimensional propagation paths. As a research direction for the future, it would be great to extend the cohesive zone model to three-dimensional structures. The introduction of large displacements would be another step forward in the development of the code.

7.1 Future works

Future works will then consider mainly two aspects, one practical (more powerful programming language) and the other one more theoretical (the extension to three-dimensional cracks).

The practical problem will therefore see the migration to a more powerful language programming, more attention to storage problems and as a final goal, the production of an effective simulation tool suitable for industry. The choice to a different programming language should also take into account the inability of normal desktop computers to carry out simulations with millions of degrees of freedom. Proper programming is necessary if parallel capabilities need to be exploited. It would be also highly desirable to further expand the capabilities of the codes to include large deformations, aiming towards the simulation of impact phenomena, in order to fully appreciate the ideas of the meshfree technology.

The present thesis has shown that it is possible to reduce the computational costs of the construction of the shape functions. The burden of the neighbour search has been alleviated through the use of the *kd*-tree algorithm. Nevertheless, it has been shown that, for a fixed number of degrees of freedom, the computational cost goes linearly with the number of Gaussian points. Gaussian points are needed to numerically evaluate the integrals in a weak form of the equations of equilibrium. The experience of the author is that a good compromise between accuracy and costs is given by a Gaussian quadrature of order 3, which means 9 Gaussian points per triangular element in two-dimensions and 27 Gaussian points per tetrahedral element in three-dimensions.

This means that the *bottle neck* is represented by the number of Gaussian points, even if with the proposed method they can be computed in a faster manner with respect to the traditional *point-wise* LU factorization. In this sense, a leap ahead would be then represented by further researching stabilization of nodal integration techniques. Nodal integration is the *Holy Grail* of the meshless methods. A truly-meshfree method that do not require a background mesh for the purposes of the integration, will then eliminate the costs due to the necessity of Gaussian points.

Appendix A

Computation of the Integral Terms in Reproducing Kernel Methods

In this appendix it is described a procedure to calculate the integrals in equation (3.41), (3.47) and (3.48), as mentioned in section 3.3.

The full description can be found in Barbieri & Meo (2009b) and Barbieri & Meo (submitted), here only the major points will be recalled. Particularly in this appendix, only the calculation of the moments will be shown.

A.1 The Window Function w and its primitives H_n

In this section are presented preliminary considerations about the window function $w(\xi)$ and its primitives. Window functions in the case of RKM could have *compact* support (for instance spline functions) or not, like Gaussian function. In both cases though, it is possible to define their primitives. Their evaluation

can be made once and for all for example through softwares capable of symbolic computations. Moreover, since primitives are infinite in number, in order to define H_n functions uniquely, a further condition need to be imposed. Defining the variable, with x' being the integration variable

$$\xi = \frac{x' - x}{\rho} \tag{A.1}$$

the functions H_n are $n+1$ antiderivatives, or primitives of the window

$$H_n(\xi) = \underbrace{\int \int \dots \int}_{n+1} w(\xi) d\xi. \tag{A.2}$$

Obviously, the following properties hold

$$\frac{dH_n}{d\xi} = H_{n-1}, \tag{A.3a}$$

$$\frac{d^{n+1}}{d\xi^{n+1}} H_n = w(\xi), \tag{A.3b}$$

$$\frac{\partial H_n}{\partial x'} = \frac{1}{\rho_x} H_{n-1}(\xi), \tag{A.3c}$$

$$\frac{\partial H_n}{\partial x} = -\frac{1}{\rho_x} H_{n-1}(\xi), \tag{A.3d}$$

$$\frac{\partial^{(n+1)}}{\partial x^{(n+1)}} H_n = -\frac{1}{\rho_x^{n+1}} (-1)^{n+1} w(\xi). \tag{A.3e}$$

Moreover, it is possible to evaluate the moments $J_k(\xi)$ of the window using these functions. Indeed, applying the recursive integration by parts

$$J_k(\xi) = \int \xi^k w(\xi) d\xi = \sum_{i=0}^{k} (-1)^i \frac{k!}{(k-i)!} \xi^{k-i} H_i(\xi), \tag{A.4}$$

for example

$$J_0(\xi) = \int \xi^0 w(\xi) d\xi = H_0(\xi), \tag{A.5a}$$

$$J_1(\xi) = \int \xi^1 w(\xi) d\xi = \xi H_0(\xi) - H_1(\xi), \tag{A.5b}$$

$$J_2(\xi) = \int \xi^2 w(\xi) d\xi = \xi^2 H_0(\xi) - 2\xi H_1(\xi) + 2H_2(\xi), \tag{A.5c}$$

$$J_3(\xi) = \int \xi^3 w(\xi) d\xi = \xi^3 H_0(\xi) - 3\xi^2 H_1(\xi) + 6\xi H_2(\xi) - 6H_3(\xi). \tag{A.5d}$$

A.1.1 Non Compact Support Kernels

An example of kernels having non-compact support is the *Gaussian function*

$$w(\xi) = e^{-\frac{1}{2}\xi^2}. \tag{A.6}$$

The particular function in equation (A.6) it is 1 for $\xi = 0$ and it rapidly decays to zero (even though is it not exactly zero) around $\xi = \pm 3$, as it can be seen from figure A.1a. Moreover, since primitives are infinite in number, in order to define H_n functions uniquely, a further condition need to be imposed. For non-compact support, the following condition is chosen

$$\lim_{x \to -\infty} H_0(\xi) = 0. \tag{A.7}$$

In this case, H_n functions are

$$H_0(\xi) = \frac{\sqrt{2\pi}}{2} \ \mathrm{erf}\left(\frac{\sqrt{2}}{2}\xi\right) + \frac{\sqrt{2\pi}}{2}, \tag{A.8}$$

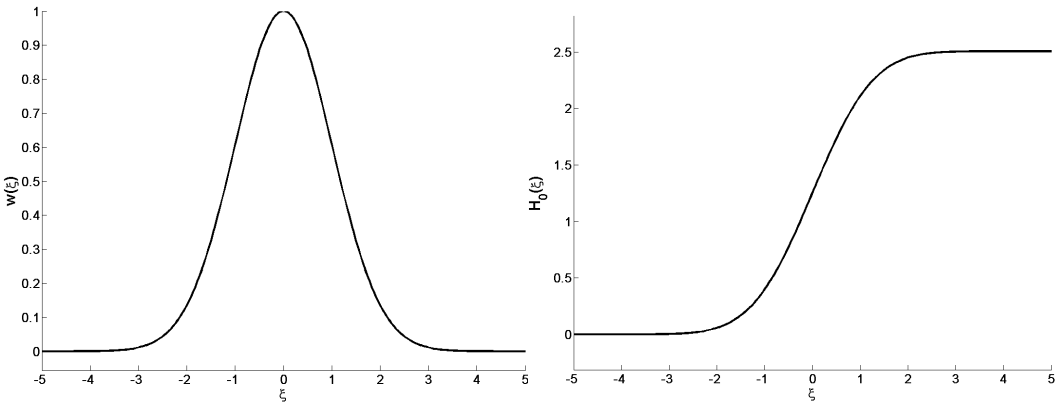
$$H_1(\xi) = \sqrt{\pi}\left(\frac{\sqrt{2}}{2}\,\xi\,\mathrm{erf}\left(\frac{\sqrt{2}}{2}\,\xi\right) + \frac{e^{-1/2\,\xi^2}}{\sqrt{\pi}}\right) + \frac{\sqrt{2\pi}}{2}\xi, \tag{A.9}$$

$$H_2(\xi) = \sqrt{\pi}\left(\frac{\sqrt{2}}{2}\left(\frac{\xi^2}{2}\mathrm{erf}\left(\frac{\sqrt{2}}{2}\xi\right) - 2\left(-\frac{\sqrt{2}\xi}{4e^{\frac{\xi^2}{2}}} + \frac{\sqrt{\pi}}{4}\mathrm{erf}\left(\frac{\sqrt{2}}{2}\xi\right)\right)\frac{1}{\sqrt{\pi}}\right) + \right.$$
$$\left. + \frac{\sqrt{2}}{2}\mathrm{erf}\left(\frac{\sqrt{2}}{2}\xi\right)\right) + \frac{\sqrt{2\pi}}{4}\xi^2, \tag{A.10}$$

where $\mathrm{erf}\,(x)$ is the *error function* defined as

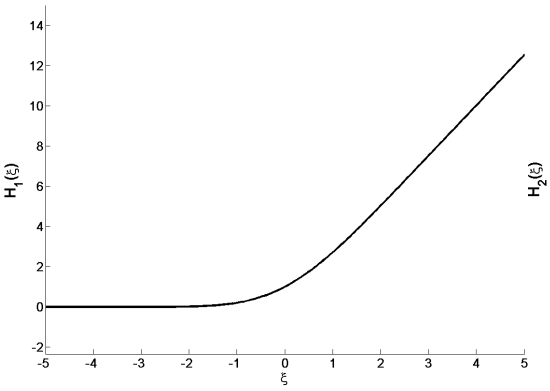
$$\mathrm{erf}\,(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\mathrm{d}t. \tag{A.11}$$

This functions are plotted in figures A.1. It can be observed that these functions (figures A.1b, A.1c and A.1d) outside $\xi = 3$ have respectively constant, linear and quadratic behavior. This means that the boundary correction terms influence only a small portion of the domain around the boundaries of measure ρ, as it is also reported in Liu *et al.* (1997a). This is more evident for the moment matrix entries depicted in figures A.3a, A.3b and A.3c.

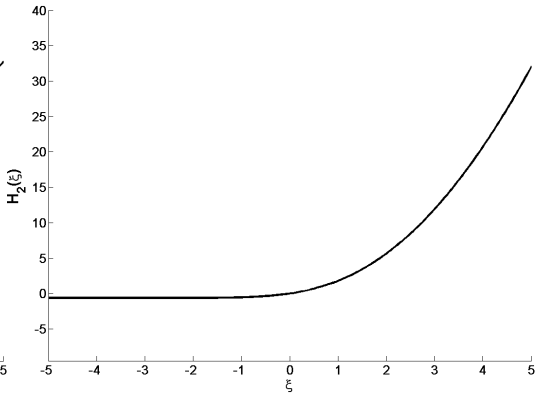


(a) Kernel Function

(b) Primitive H_0



(c) Primitive H_1



(d) Primitive H_2

Figure A.1: Non-Compact Support Kernel Functions: Gaussian Function

A.1.2 Compact Support Kernels

An example of compact support kernel is the $2k^{th}$ order spline which can be rewritten using the *Heaviside* function $\mathcal{H}$ as

$$w(\xi) = (1-\xi^2)^k \left[\mathcal{H}(\xi+1) - \mathcal{H}(\xi-1)\right]. \tag{A.12}$$

For compact support functions, the condition on the primitive has been chosen as

$$H_0(\xi = -1) = 0. \tag{A.13}$$

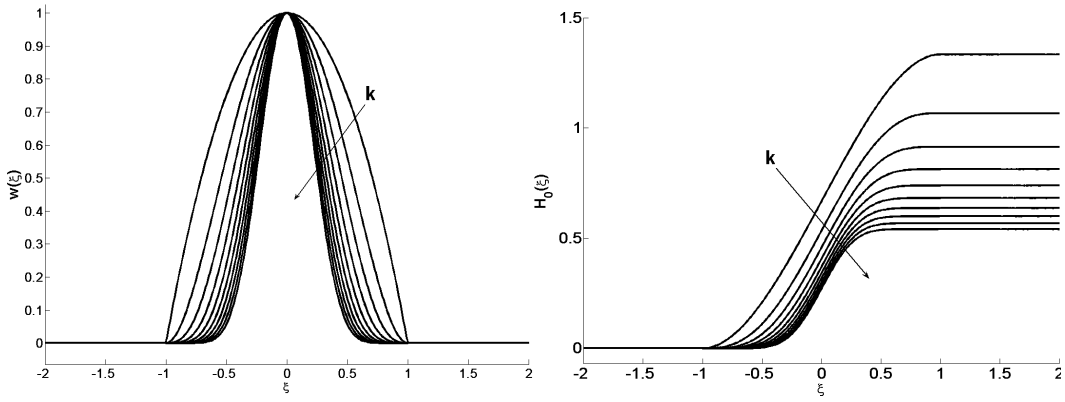
Thus, naming as ${}_2F_1(a,b;c;z)$ the *classical standard hypergeometric* series

$$\begin{aligned} H_0(\xi) = \xi\ {}_2F_1(\frac{1}{2},-k;\frac{3}{2};\xi^2)\left[\mathcal{H}(\xi+1) - \mathcal{H}(\xi-1)\right] + \\ +\ {}_2F_1(\frac{1}{2},-k;\frac{3}{2};1)\left[\mathcal{H}(\xi+1) + \mathcal{H}(\xi-1)\right], \end{aligned} \tag{A.14}$$

$$\begin{aligned} H_1(\xi) = \left[{}_2F_1(\frac{1}{2},-k;\frac{3}{2};\xi^2)(\xi-1) - \frac{1}{2(k+1)}\ {}_2F_1(-\frac{1}{2},-k-1;\frac{1}{2};\xi^2)\right]\mathcal{H}(\xi-1) + \\ + \left[\frac{1}{2(k+1)}\ {}_2F_1(-\frac{1}{2},-k-1;\frac{1}{2};\xi^2) +\ {}_2F_1(\frac{1}{2},-k;\frac{3}{2};1)(\xi+1) + \right. \\ \left. -\frac{1}{2(k+1)}\ {}_2F_1(-\frac{1}{2},-k-1;\frac{1}{2};1)\right]\mathcal{H}(\xi+1). \end{aligned} \tag{A.15}$$

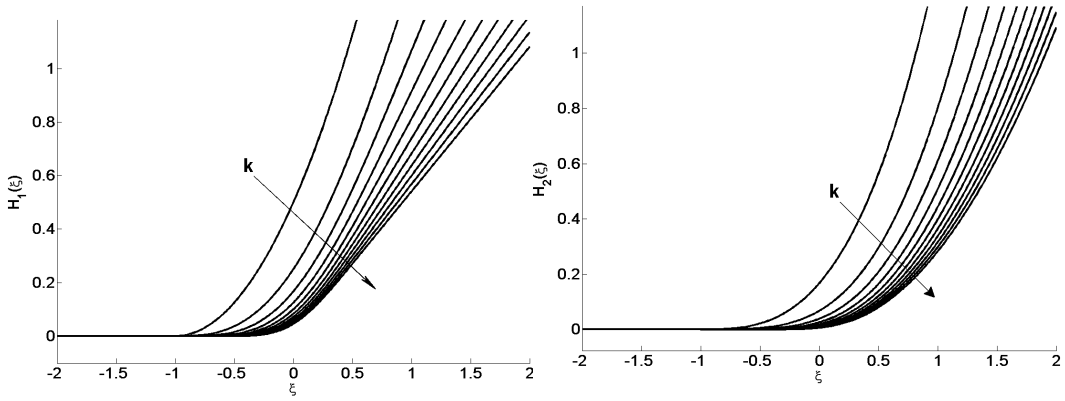
This functions are drawn in figures A.2 for different degree k of spline. It can be noted again the influence of the correction terms outside the support i.e. $\xi = 1$. Moreover, it is useful to define the primitives $I_k(\xi)$ of the moments $J_k(\xi)$. These functions are used in the computation of moments for three-dimensional domains.

$$I_k(\xi) = \int J_k(\xi)d\xi = \sum_{i=0}^{k}(-1)^i(i+1)\frac{k!}{(k-i)!}\xi^{k-i}H_{i+1}(\xi) \tag{A.16}$$



(a) Kernel Function

(b) Primitive H_0



(c) Primitive H_1

(d) Primitive H_2

Figure A.2: Compact Support Kernel Functions: $2k-th$ order spline

for example

$$I_0(\xi) = \int J_0(\xi)d\xi = H_1(\xi), \tag{A.17a}$$

$$I_1(\xi) = \int J_1(\xi)d\xi = \xi H_1(\xi) - 2H_2(\xi), \tag{A.17b}$$

$$I_2(\xi) = \int J_2(\xi)d\xi = \xi^2 H_1(\xi) - 4\xi H_2(\xi) + 6H_3(\xi), \tag{A.17c}$$

$$I_3(\xi) = \int J_3(\xi)d\xi = \xi^3 H_1(\xi) - 6\xi^2 H_2(\xi) + 18\xi H_3(\xi) - 24H_4(\xi). \tag{A.17d}$$

A.2 Moments Matrix in One Dimension

Using the H_n functions, it is immediate to evaluate the entries in the moments matrix.

If $\Omega = [0, L]$, then the generic entry M_{ij} of the moments matrix is

$$M_{ij}(x) = \int_0^L \xi^{i+j} w(\xi)dx' = \rho_x \int_{\frac{-x}{\rho_x}}^{\frac{L-x}{\rho_x}} \xi^{i+j} w(\xi)\mathrm{d}\xi. \tag{A.18}$$

More generally:

$$\int_{-\frac{x}{\rho_x}}^{\frac{L-x}{\rho_x}} \xi^k w(\xi)\mathrm{d}\xi = J_k\left(\frac{L-x}{\rho_x}\right) - J_k\left(-\frac{x}{\rho_x}\right). \tag{A.19}$$

Therefore, from equation (A.4), it follows

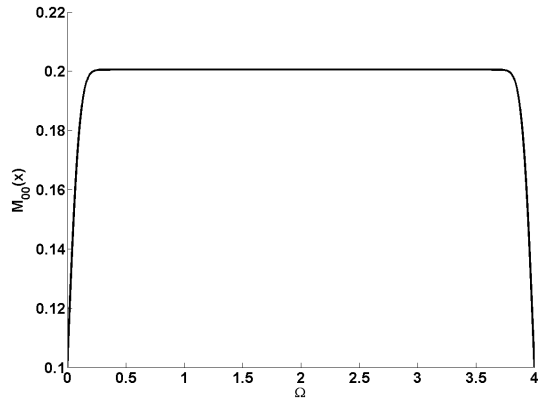
$$M_{ij}(x) = \rho_x \left[\sum_{l=0}^{i+j}(-1)^l \frac{(i+j)!}{(i+j-l)!}\xi^{i+j-l} H_l(\xi)\right]_{\frac{-x}{\rho_x}}^{\frac{L-x}{\rho_x}}. \tag{A.20}$$

In figures A.4a and A.4b correction functions are plotted for basis functions $\mathbf{p}^T(x) = [1 \;\; x]$. Indeed correction terms are given by $\mathbf{p}^T(0)\mathbf{M}(\mathbf{x})^{-1}$ that is

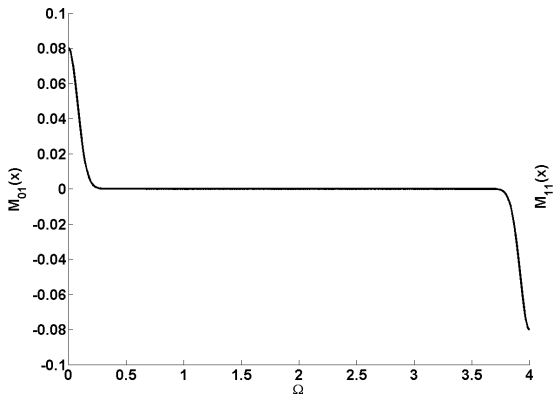
$$\mathbf{p}^T(0)\mathbf{M}(x)^{-1} = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}\mathbf{M}_{11}^{-1}(x) & \mathbf{M}_{12}^{-1}(x) \\ \mathbf{M}_{12}^{-1}(x) & \mathbf{M}_{22}^{-1}(x)\end{bmatrix} = \begin{bmatrix}\mathbf{M}_{11}^{-1}(x) & \mathbf{M}_{12}^{-1}(x)\end{bmatrix} \tag{A.21}$$

where, using symbolic evaluation of the inverse of $\mathbf{M}(x)$

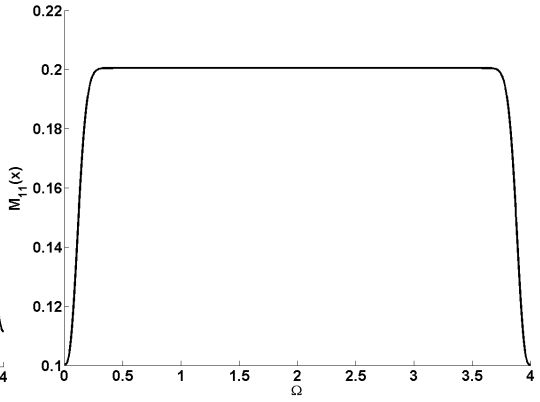
$$\mathbf{M}_{11}^{-1}(x) = \frac{M_{22}(x)}{M_{11}(x)M_{22}(x) - M_{12}(x)^2}, \tag{A.22}$$



(a) $M_{00}(x)$

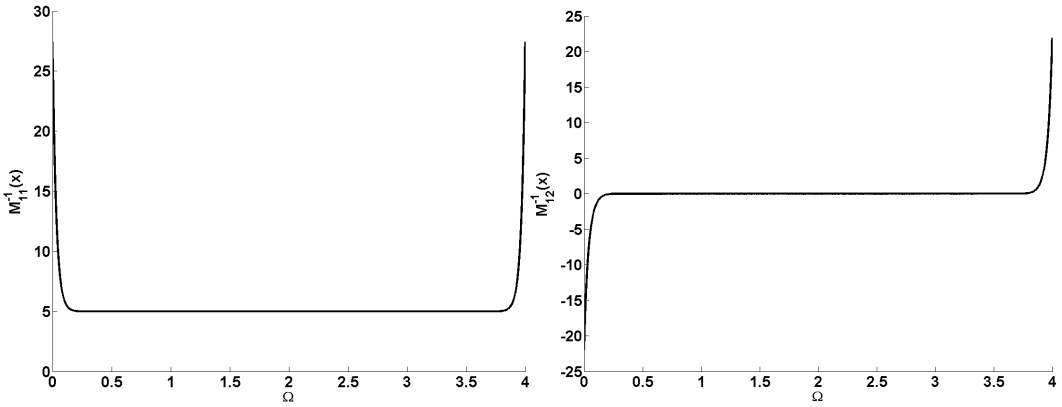


(b) $M_{01}(x) = M_{10}(x)$



(c) $M_{11}(x) = M_{20}(x) = M_{02}(x)$

Figure A.3: Moments Matrix in One Dimension



(a) $M_{11}^{-1}(x)$ (b) $M_{12}^{-1}(x)$

Figure A.4: Correction Factors in One Dimension for basis function $\mathbf{p}^T(x) = [1 \quad x]$

$$\mathbf{M}_{12}^{-1}(x) = -\frac{M_{12}(x)}{M_{11}(x)M_{22}(x) - M_{12}(x)^2}. \tag{A.23}$$

The calculation of $\dfrac{\partial \mathbf{M}(\mathbf{x})^{-1}}{\partial x_i}$ and $\dfrac{\partial \Lambda(\mathbf{x})}{\partial x}$ is necessary to get the derivatives of the shape functions. First derivatives of the moment matrix can be calculated promptly. In fact

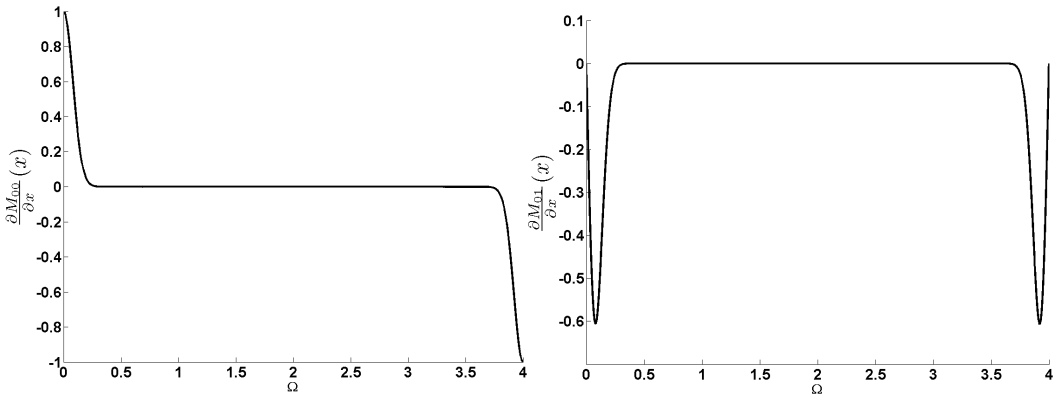
$$\frac{\partial \mathbf{M}(\mathbf{x})^{-1}}{\partial x} = -\mathbf{M}(\mathbf{x})^{-1}\frac{\partial \mathbf{M}(\mathbf{x})}{\partial x}\mathbf{M}(\mathbf{x})^{-1}. \tag{A.24}$$

Regarding the moments matrix

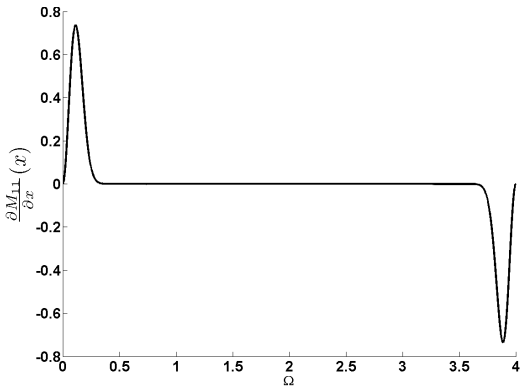
$$\begin{aligned}\frac{\partial \mathbf{M}_{ij}(x)}{\partial x} = \int_0^L \frac{\partial}{\partial x}\left(\xi^{i+j} w(\xi)\right) dx' = \int_0^L \frac{\partial}{\partial \xi}\left(\xi^{i+j} w(\xi)\right)\frac{\partial \xi}{\partial x} dx' = \\ = -\int_{\frac{-x}{\rho_x}}^{\frac{L-x}{\rho_x}} \frac{\partial}{\partial \xi}\left(\xi^{i+j} w(\xi)\right) d\xi \end{aligned} \tag{A.25}$$

then

$$\frac{\partial \mathbf{M}_{ij}(x)}{\partial x} = \left(-\frac{x}{\rho_x}\right)^{i+j} w\left(-\frac{x}{\rho_x}\right) - \left(\frac{L-x}{\rho_x}\right)^{i+j} w\left(\frac{L-x}{\rho_x}\right). \tag{A.26}$$

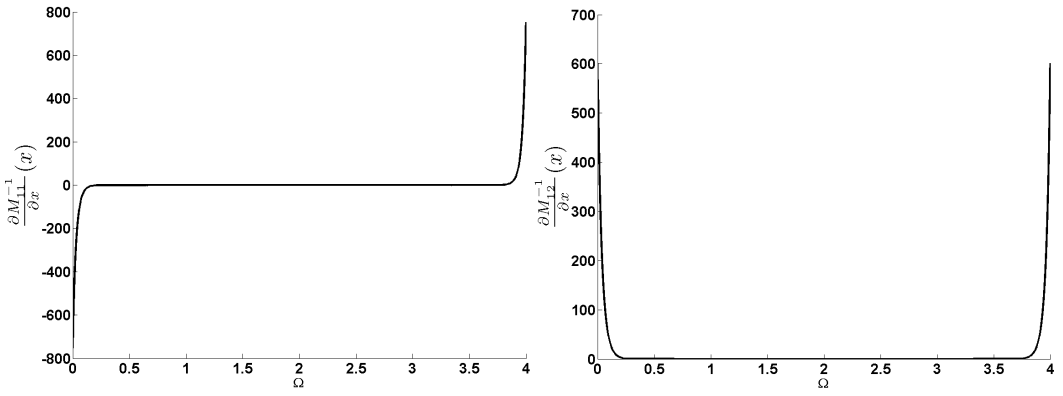


(a) $\frac{\partial M_{00}}{\partial x}(x)$ (b) $\frac{\partial M_{01}}{\partial x}(x)$



(c) $\frac{\partial M_{11}(x)}{\partial x}$

Figure A.5: First Derivative of Moments Matrix in One Dimension



(a) $\dfrac{\partial M_{11}^{-1}}{\partial x}(x)$ (b) $\dfrac{\partial M_{12}^{-1}}{\partial x}(x)$

Figure A.6: First Derivative of Correction Factors in One Dimension for basis function $\mathbf{p}^T(x) = [1 \;\; x]$

A.3 Two-Dimensional Domains

It is possible to generalize to two and three dimensions the procedure described in the previous section.

A *tensor product* kernel of the type

$$w\left(\frac{\mathbf{x}' - \mathbf{x}}{\rho}\right) = w\left(\frac{x' - x}{\rho_x}\right) w\left(\frac{y' - y}{\rho_y}\right) \tag{A.27}$$

will be used in this section.

The generic entry for the moment matrix is

$$\mathbf{M}_{ij}(x, y) = \iint_{\Omega'} \xi^i \eta^j w(\xi) w(\eta) dx' dy' \tag{A.28}$$

where

$$\eta = \frac{y' - y}{\rho_y}, \tag{A.29}$$

$$\int_{\Omega_{n\xi}} \xi^k \eta^l w(\xi) w(\eta) \mathrm{d}\xi \mathrm{d}\eta = \oint_{\partial\Omega_{n\xi}} \mathbf{F}^{kl}(\xi, \eta) \cdot \mathrm{d}\mathbf{n} \tag{A.30}$$

where $\mathrm{d}\mathbf{n}$ is the infinitesimal normal vector to $\partial\Omega_{n\xi}$ pointing outwards (figure A.7) and $\mathbf{F}^{kl}$ a vectorial field resulting from

$$\nabla \cdot \mathbf{F}^{kl}(\xi, \eta) = \frac{\partial F_\xi^{kl}}{\partial \xi} + \frac{\partial F_\eta^{kl}}{\partial \eta} = \xi^k \eta^l w(\xi) w(\eta). \tag{A.31}$$

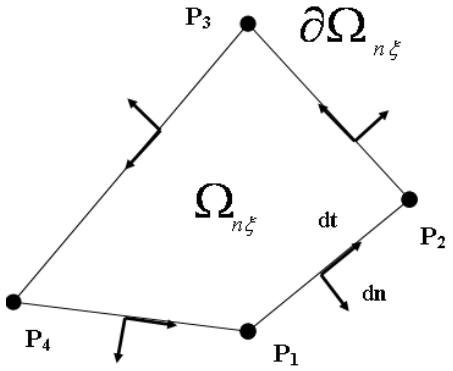


Figure A.7: Four Nodes Quadrilateral element

No boundary conditions are required to solve (A.31), therefore the following equations satisfy (A.31)

$$\begin{cases} \dfrac{\partial F_{\xi}^{kl}}{\partial \xi} = \frac{1}{2}\xi^k \eta^l w(\xi) w(\eta) \\ \dfrac{\partial F_{\eta}^{kl}}{\partial \eta} = \frac{1}{2}\xi^k \eta^l w(\xi) w(\eta) \end{cases} . \tag{A.32}$$

Thus

$$F_{\xi}^{kl}(\xi,\eta) = \int \frac{1}{2}\xi^k \eta^l w(\xi) w(\eta) \mathrm{d}\xi = \frac{1}{2} J_k(\xi) \eta^l w(\eta), \tag{A.33a}$$

$$F_{\eta}^{kl}(\xi,\eta) = \int \frac{1}{2}\xi^k \eta^l w(\xi) w(\eta) \mathrm{d}\eta = \frac{1}{2} \xi^k w(\xi) J_l(\eta). \tag{A.33b}$$

Moreover, formula (A.31) facilitates the computation of the above mentioned integrals if Ω has one or more holes. In this case the circular integral needs to be carried out on two or more boundaries, since domains can be seen as a difference among separated domains (figure A.8).

In fact, according to equation (A.30),

$$\oint_{\partial\Omega_{n\xi}} \mathbf{F}^{kl}(\xi,\eta) \cdot \mathrm{d}\mathbf{n} = \oint_{\partial\Omega_{n\xi^e}} \mathbf{F}^{kl}(\xi,\eta) \cdot \mathrm{d}\mathbf{n} - \oint_{\partial\Omega_{n\xi^h}} \mathbf{F}^{kl}(\xi,\eta) \cdot \mathrm{d}\mathbf{n} \tag{A.34}$$

where $\partial\Omega_{n\xi^e}$ is the external boundary and $\partial\Omega_{n\xi^h}$ is the hole boundary. If n_h is the number of holes, one just needs to subtract to equation (A.34) all the terms related to the holes $\partial\Omega_{n\xi^{h_i}} \quad i = 1, \ldots, n_h$.

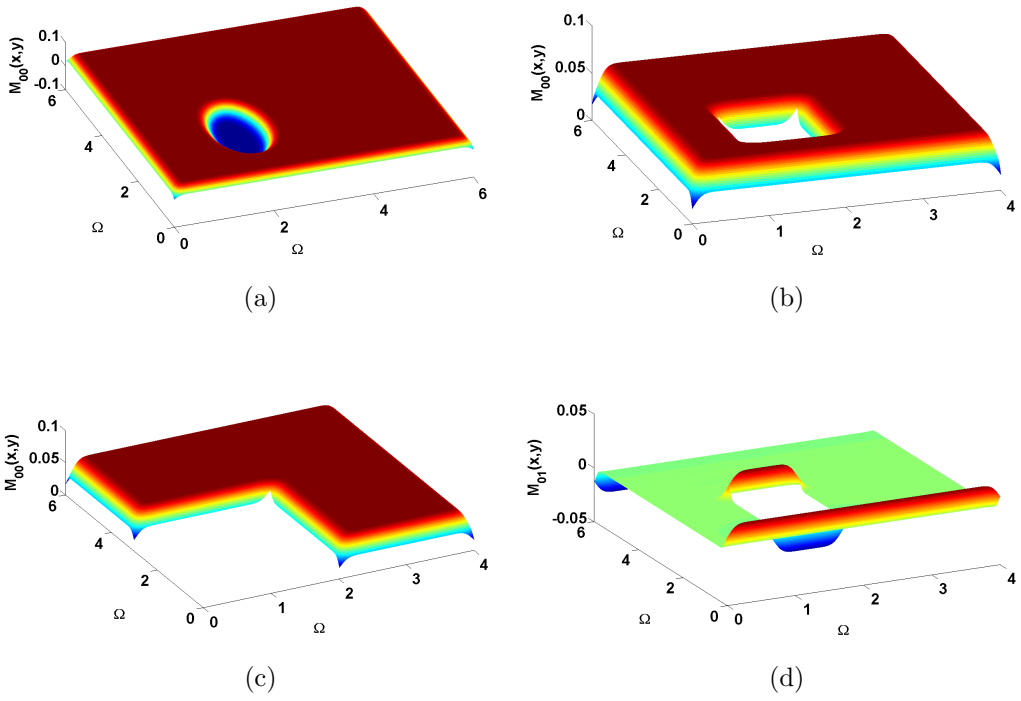


Figure A.8: Examples of moment matrix entries in two dimensions

A.4 Three-Dimensional Domains

The same approach applies to three-dimensional domains, where the generic integral in equation (A.30) is extended as

$$\int_{\Omega_{n\xi}} \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta) \mathrm{d}\xi \mathrm{d}\eta = \oiint_{\partial\Omega_{n\xi}} \mathbf{F}^{klm}(\xi, \eta, \zeta) \cdot \mathrm{d}\mathbf{n} \tag{A.35}$$

where $\mathrm{d}\mathbf{n}$ is the infinitesimal normal vector to $\partial\Omega$ pointing outwards and $\mathbf{F}^{klm}$ a vectorial field resulting from

$$\nabla \cdot \mathbf{F}^{klm}(\xi, \eta, \zeta) = \frac{\partial F_\xi^{klm}}{\partial \xi} + \frac{\partial F_\eta^{klm}}{\partial \eta} + \frac{\partial F_\zeta^{klm}}{\partial \eta} = \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta). \tag{A.36}$$

Thus,

$$F_\xi^{klm}(\xi, \eta, \zeta) = \int \frac{1}{3} \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta) \mathrm{d}\xi = \frac{1}{3} J_k(\xi) \eta^l w(\eta) \zeta^m w(\zeta), \tag{A.37a}$$

$$F_\eta^{klm}(\xi, \eta, \zeta) = \int \frac{1}{3} \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta) \mathrm{d}\eta = \frac{1}{3} \xi^k w(\xi) J_l(\eta) \zeta^m w(\zeta), \tag{A.37b}$$

$$F_\zeta^{klm}(\xi, \eta, \zeta) = \int \frac{1}{3} \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta) \mathrm{d}\zeta = \frac{1}{3} \xi^k w(\xi) \eta^l w(\eta) J_m(\zeta). \tag{A.37c}$$

A further reduction of integral (A.35) can be made if $\Omega_{n\xi}$ is part of a *polygon subdivision* as defined in Li *et al.* (2004), here recalled

Polygon subdivision A partition $\mathcal{F}_n = \{\Omega_1,\ \Omega_2\ \ldots \Omega_n\}$ is a polygon subdivision if

1. $\Omega_i\ i \in 1, \ldots, n$ is a polygon;
2. $\Omega_i \cap \Omega_j = \emptyset\ i \neq j$;
3. $\bigcup_{i=1}^n \overline{\Omega}_i = \overline{\Omega}$;
4. if $\overline{\Omega}_i \cap \overline{\Omega}_j\ i \neq j$ consists of exactly one point, then it is the common vertex of the two polygons;
5. if $\overline{\Omega}_i \cap \overline{\Omega}_j\ i \neq j$ consists of more than one point, then $\overline{\Omega}_i \cap \overline{\Omega}_j$ is either the common surface or the common edge.

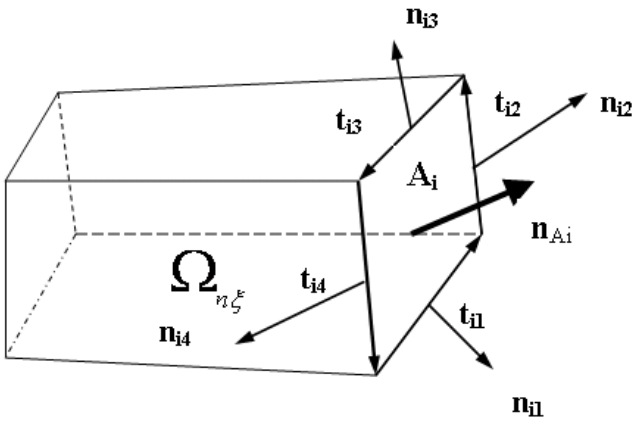


Figure A.9: Three-dimensional polygonal subdivision

In this case $\Omega_{n\xi}$ is a *polygon*, therefore $\partial\Omega_{n\xi}$ is the union of n_f *faces* A_i that are polygon themselves (figure A.9).

$$\partial\Omega_{n\xi} = \bigcup_{i=1}^{n_f} A_i. \tag{A.38}$$

Let $\mathbf{n}_{Ai}$ the (constant) normal vector to A_i Therefore

$$\oiint_{\partial\Omega_{n\xi}} \mathbf{F}^{klm} \cdot \mathrm{d}\mathbf{n} = \sum_{i=1}^{n_f} \mathbf{n}_{Ai}^T \iint_{A_i} \mathbf{F}^{klm} \mathrm{d}A_i. \tag{A.39}$$

Again the Gauss' theorem can be used for each face

$$\iint_{A_i} \mathbf{F}^{klm} \mathrm{d}A_i = \oint_{\partial A_i} \mathbf{G}^{klm}\ \mathbf{n} \mathrm{d}s \tag{A.40}$$

where $\mathbf{n}$ is the *constant* normal vector of the common edge, but which belongs to the face A_i and uniquely defined as

$$\mathbf{n}_{ij} = \mathbf{t}_{ij} \times \mathbf{n}_{Ai} \tag{A.41}$$

where $\mathbf{t}_{ij}$ is the (constant) tangent vector of the *j-th* edge that belongs to the *i-th* face. With the definition (A.41) the normal vector in integral (A.40) points outwards ∂A_i. The matrix in integral (A.40) is given by

$$\nabla \cdot \mathbf{G}^{klm}(\xi, \eta, \zeta) = \mathbf{F}^{klm}(\xi, \eta, \zeta) \tag{A.42}$$

that is

$$\frac{\partial G_{\xi\xi}^{klm}}{\partial \xi} + \frac{\partial G_{\xi\eta}^{klm}}{\partial \eta} + \frac{\partial G_{\xi\zeta}^{klm}}{\partial \zeta} = F_{\xi}^{klm}, \tag{A.43a}$$

$$\frac{\partial G_{\eta\xi}^{klm}}{\partial \xi} + \frac{\partial G_{\eta\eta}^{klm}}{\partial \eta} + \frac{\partial G_{\eta\zeta}^{klm}}{\partial \zeta} = F_{\eta}^{klm}, \tag{A.43b}$$

$$\frac{\partial G_{\zeta\xi}^{klm}}{\partial \xi} + \frac{\partial G_{\zeta\eta}^{klm}}{\partial \eta} + \frac{\partial G_{\zeta\zeta}^{klm}}{\partial \zeta} = F_{\zeta}^{klm}. \tag{A.43c}$$

Equations (A.43) can be evaluated as usual

$$G_{\xi\xi}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\xi}^{klm} \mathrm{d}\xi = \frac{1}{9} I_k(\xi)\eta^l w(\eta)\zeta^m w(\zeta), \tag{A.44a}$$

$$G_{\xi\eta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\xi}^{klm} \mathrm{d}\eta = \frac{1}{9} J_k(\xi) J_l(\eta)\zeta^m w(\zeta), \tag{A.44b}$$

$$G_{\xi\zeta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\xi}^{klm} \mathrm{d}\zeta = \frac{1}{9} J_k(\xi)\eta^l w(\eta) J_m(\zeta), \tag{A.44c}$$

$$G_{\eta\xi}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\eta}^{klm} \mathrm{d}\xi = \frac{1}{9} J_k(\xi) J_l(\eta)\zeta^m w(\zeta) = G_{\xi\eta}^{klm}(\xi,\eta,\zeta), \tag{A.44d}$$

$$G_{\eta\eta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\eta}^{klm} \mathrm{d}\eta = \frac{1}{9} \xi^k w(\xi) I_l(\eta)\zeta^m w(\zeta), \tag{A.44e}$$

$$G_{\eta\zeta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\eta}^{klm} \mathrm{d}\zeta = \frac{1}{9} \xi^k w(\xi) J_l(\eta) J_m(\zeta), \tag{A.44f}$$

$$G_{\zeta\xi}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\zeta}^{klm} \mathrm{d}\xi = \frac{1}{9} J_k(\xi)\eta^l w(\eta) J_m(\zeta) = G_{\xi\zeta}^{klm}(\xi,\eta,\zeta), \tag{A.44g}$$

$$G_{\zeta\eta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\zeta}^{klm} \mathrm{d}\eta = \frac{1}{9} \xi^k w(\xi) J_l(\eta) J_m(\zeta) = G_{\eta\zeta}^{klm}(\xi,\eta,\zeta), \tag{A.44h}$$

$$G_{\zeta\zeta}^{klm}(\xi,\eta,\zeta) = \int \frac{1}{3} F_{\zeta}^{klm} \mathrm{d}\zeta = \frac{1}{9} \xi^k w(\xi)\eta^l w(\eta) I_m(\zeta). \tag{A.44i}$$

$$\tag{A.44j}$$

Therefore if $\Omega_{n\xi}$ is a polygon, (for example like in FE discretization)

$$\int_{\Omega_{n\xi}} \xi^k \eta^l \zeta^m w(\xi) w(\eta) w(\zeta) \mathrm{d}\xi \mathrm{d}\eta = \sum_{i=1}^{n_f} \mathbf{n}_{Ai}^T \sum_{j=1}^{n_s} \left(\int_{e_{ij}} \mathbf{G}^{klm} \mathrm{ds} \right) \mathbf{n}_{ij} \tag{A.45}$$

where n_s is the number of segments of the face and e_{ij} is the *j-th* edge that belongs to the *i-th* face with normal vector $\mathbf{n}_{ij}$. Thus it can be concluded that every

integral for two and three-dimensional polygon subdivision is then reformulated as a line integral for which an explicit expression can be provided.

A.5 Explicit Expression for Line Integrals

In order to calculate line integrals parametrization of the boundaries must be provided.

For polygon subdivision, these lines are actually segments that are easy to parametrize. If the orientated segment has starting point $P_1 = (x_1, y_1)$ and end point $P_2 = (x_2, y_2)$, then

$$\begin{cases} \xi(t) = h(x) + \alpha t & , \\ \eta(t) = q(y) + \beta t & t \in [0,1] \end{cases} \tag{A.46}$$

with $h = \frac{x_1 - x}{\rho_x}$, $q = \frac{y_1 - y}{\rho_y}$, $\alpha = \frac{x_2 - x_1}{\rho_x}$ and $\beta = \frac{y_2 - y_1}{\rho_y}$ and normal vector $\mathbf{n}^T = \begin{bmatrix} \beta & -\alpha \end{bmatrix}$.

These line integrals can be evaluated explicitly through symbolic manipulation.

As an example

$$\int_{P_1}^{P_2} F_\xi^{00}(\xi,\eta)\mathrm{d}\eta - F_\eta^{00}(\xi,\eta)\mathrm{d}\xi = \frac{\beta}{2}\int_0^1 H_0(h+\alpha t)w(q+\beta t)\ \mathrm{d}t + \\ - \frac{\alpha}{2}\int_0^1 w(h+\alpha t)H_0(q+\beta t)\ \mathrm{d}t. \tag{A.47}$$

If either $\alpha = 0$ (vertical segment) or $\beta = 0$ (horizontal segment), the integral (A.47) is trivial since if $\alpha = 0$

$$\frac{\beta}{2}\int_0^1 H_0(h)w(q+\beta t)\ \mathrm{d}t = \frac{H_0(h)}{2}\int_0^1 w(q+\beta t)\ \mathrm{d}(q+\beta t) = \\ = \frac{H_0(h)}{2}\left[H_0(q+\beta) - H_0(q)\right] \tag{A.48}$$

whereas if $\beta = 0$

$$\frac{\alpha}{2}\int_0^1 H_0(q)w(h+\alpha t)\ \mathrm{d}t = \frac{H_0(q)}{2}\int_0^1 w(h+\alpha t)\ \mathrm{d}(h+\alpha t) = \\ = \frac{H_0(q)}{2}\left[H_0(h+\alpha) - H_0(h)\right]. \quad \text{(A.49)}$$

If both $\alpha \neq 0$ and $\beta \neq 0$ the expression becomes much more complicated, even with simple spline window functions.

Defining the window function as

$$w(\xi) = f(\xi)\left[\mathcal{H}(\xi+1) - \mathcal{H}(\xi-1)\right] \quad \text{(A.50)}$$

for example in (A.12)

$$f(\xi) = (1-\xi^2)^k \quad \text{(A.51)}$$

and defining the primitive of $f(\xi)$

$$K_0(\xi) = \int_{-1}^{\xi} f(\xi)\mathrm{d}\xi \quad \text{(A.52)}$$

function $H_0(\xi)$ can be written as

$$H_0(\xi) = K_0(\xi)\left[\mathcal{H}(\xi+1) - \mathcal{H}(\xi-1)\right] + K_0(1)\mathcal{H}(\xi-1). \quad \text{(A.53)}$$

It has been experienced by the authors that symbolic softwares cannot provide *directly* a closed form of the integral (A.47). This is mostly caused by the presence of the *Heaviside* functions that can easily halt the calculation. If an explicit form is desired, some preliminary manipulation is needed. Firstly, it can be noted from equation (A.47) that the second term of the right hand side is exactly the same of the first one provided that α and h are interchanged with β and q. Therefore, it suffices to calculate only the first integral in (A.47). Such integral can be expanded in the following form using equations (A.50), (A.52) and (A.53)

$$\int_0^1 H_0(h+\alpha t)w(q+\beta t)\,\mathrm{d}t = \int_0^1 K_0(h+at)f(q+bt)\,\mathcal{H}(at+h+1)\mathcal{H}(bt+q+1)\mathrm{d}t+$$
$$-\int_0^1 K_0(h+at)f(q+bt)\,\mathcal{H}(at+h-1)\mathcal{H}(bt+q+1)\mathrm{d}t+$$
$$-\int_0^1 K_0(h+at)f(q+bt)\,\mathcal{H}(at+h+1)\mathcal{H}(bt+q-1)\mathrm{d}t$$
$$+\int_0^1 K_0(h+at)f(q+bt)\,\mathcal{H}(at+h-1)\mathcal{H}(bt+q-1)\mathrm{d}t+$$
$$+\ K_0(1)\int_0^1 f(q+bt)\,\mathcal{H}(at+h-1)\mathcal{H}(bt+q+1)\mathrm{d}t+$$
$$-K_0(1)\int_0^1 f(q+bt)\,\mathcal{H}(at+h-1)\mathcal{H}(bt+q-1)\mathrm{d}t. \quad \text{(A.54)}$$

Considering that generally for $m = h+1, h-1$

$$\mathcal{H}(at+m) = \mathcal{H}(\alpha)\mathcal{H}\left(t+\frac{m}{\alpha}\right) + \mathcal{H}(-\alpha)\mathcal{H}\left(-\left(t+\frac{m}{\alpha}\right)\right) =$$
$$= \left(\mathcal{H}(\alpha)-\mathcal{H}(-\alpha)\right)\mathcal{H}\left(t+\frac{m}{\alpha}\right) + \mathcal{H}(-\alpha). \quad \text{(A.55)}$$

and for $p = q+1, q-1$

$$\mathcal{H}(\alpha t+m)\mathcal{H}(\beta t+p) =$$
$$= \left(\mathcal{H}(\alpha)-\mathcal{H}(-\alpha)\right)\left(\mathcal{H}(\beta)-\mathcal{H}(-\beta)\right)\mathcal{H}\left(t+\frac{m}{\alpha}\right)\mathcal{H}\left(t+\frac{p}{\beta}\right)+$$
$$\mathcal{H}(-\beta)(\mathcal{H}(\alpha)-\mathcal{H}(-\alpha))\mathcal{H}\left(t+\frac{m}{\alpha}\right) + \mathcal{H}(-\alpha)(\mathcal{H}(\beta)-\mathcal{H}(-\beta))\mathcal{H}\left(t+\frac{p}{\beta}\right)+$$
$$+\,\mathcal{H}(-\alpha)\mathcal{H}(-\beta) \quad \text{(A.56)}$$

also considering that if spline window functions are used, the functions $K_0(h+at)f(q+bt)$ and $f(q+bt)$ can be re-grouped in polynomial form as

$$K_0(h+at)f(q+bt) = \mathbf{C}(\alpha,\beta,h(x),q(y))^T\mathbf{T} \quad \text{(A.57)}$$

where $\mathbf{T}$ is a vector of monomial powers of t

$$\mathbf{T} = \begin{bmatrix}1 & t & t^2 & \cdots & t^n\end{bmatrix} \quad \text{(A.58)}$$

and $\mathbf{C}(\alpha,\beta,h(x),q(y))^T$ is a vector of coefficients that does not depend on the variable of integration t. Therefore the generic term that appears in integral (A.54) can be re-written as

$$\begin{aligned}\int_0^1 K_0(h+\alpha t)f(q+\beta t)\mathcal{H}(\alpha t+m)\mathcal{H}(\beta t+p) = \\ (\mathcal{H}(\alpha)-\mathcal{H}(-\alpha))\,(\mathcal{H}(\beta)-\mathcal{H}(-\beta))\,\mathbf{C}^T\int_0^1 \mathbf{T}\mathcal{H}\left(t+\frac{m}{\alpha}\right)\mathcal{H}\left(t+\frac{p}{\beta}\right)\mathrm{d}t+ \\ +\mathcal{H}(-\beta)(\mathcal{H}(\alpha)-\mathcal{H}(-\alpha))\mathbf{C}^T\int_0^1 \mathbf{T}\mathcal{H}\left(t+\frac{m}{\alpha}\right)\mathrm{d}t+ \\ +\mathbf{C}^T\mathcal{H}(-\alpha)(\mathcal{H}(\beta)-\mathcal{H}(-\beta))\int_0^1 \mathbf{T}\mathcal{H}\left(t+\frac{p}{\beta}\right)\mathrm{d}t+ \\ +\mathcal{H}(-\alpha)\mathcal{H}(-\beta)\mathbf{C}^T\int_0^1 \mathbf{T}\mathrm{d}t. \quad \text{(A.59)}\end{aligned}$$

therefore one only needs to evaluate explicitly the integrals

$$\begin{aligned}I_{1n}(\alpha,\beta,n,m(x),p(y)) = \int_0^1 t^n\mathcal{H}\left(t+\frac{m}{\alpha}\right)\mathcal{H}\left(t+\frac{p}{\beta}\right)\mathrm{d}t = \\ \frac{1}{\alpha\beta(n+1)}\left[\mathcal{H}\left(\frac{m}{\alpha}+1\right)\left(\left(bm\left(-\frac{m}{\alpha}\right)^n-\alpha p\left(-\frac{p}{\beta}\right)^n\right)\mathcal{H}\left(-\frac{m}{\alpha}+\frac{p}{\beta}\right)+\right.\right. \\ \left.\left.+\alpha\left(p\left(-\frac{p}{\beta}\right)^n+\beta\right)\mathcal{H}\left(\frac{p}{\beta}+1\right)\right)-\left(\left(\beta m\left(-\frac{m}{\alpha}\right)^n-\alpha p\left(-\frac{p}{\beta}\right)^n\right)\times\right. \\ \left.\times\mathcal{H}\left(-\frac{m}{\alpha}+\frac{p}{\beta}\right)+\alpha\mathcal{H}\left(\frac{p}{\beta}\right)p\left(-\frac{p}{\beta}\right)^n\right)\mathcal{H}\left(\frac{m}{\alpha}\right)\Bigg], \quad \text{(A.60)}\end{aligned}$$

$$\begin{aligned}I_{2n}(\alpha,n,m(x)) = \int_0^1 t^n\mathcal{H}\left(t+\frac{m}{\alpha}\right)\mathrm{d}t = \frac{1}{\alpha(n+1)}\left(\mathcal{H}\left(1+\frac{m}{\alpha}\right)m\left(-\frac{m}{\alpha}\right)^n+\right. \\ \left.+\mathcal{H}\left(1+\frac{m}{\alpha}\right)\alpha-\mathcal{H}\left(\frac{m}{\alpha}\right)m\left(-\frac{m}{\alpha}\right)^n\right), \quad \text{(A.61)}\end{aligned}$$

$$\int_0^1 t^n\mathrm{d}t = \frac{1}{n+1} \quad \text{(A.62)}$$

with $n\in\mathcal{N}_0$ The same method can be used for the remaining terms in (A.54).

This approach can be repeated for all the vectorial fields $\mathbf{F}^{klm}$ and tensorial $\mathbf{G}^{klm}$ and explicit integrals can be obtained with a software capable of symbolic manipulations.

Derivatives can be readily calculated as well since from equation (A.57)

$$\frac{\partial \mathbf{C}}{\partial x}(\alpha, \beta, h(x), q(y)) = -\frac{1}{\rho_x}\frac{\partial \mathbf{C}(\alpha, \beta, h, q)}{\partial h}, \tag{A.63a}$$

$$\frac{\partial \mathbf{C}}{\partial y}(\alpha, \beta, h(x), q(y)) = -\frac{1}{\rho_y}\frac{\partial \mathbf{C}(\alpha, \beta, h, q)}{\partial q}. \tag{A.63b}$$

The right hand side of equations (A.63a) is easy to evaluate since $\mathbf{C}$ is a vector of polynomial functions in $h(x)$ and $q(y)$.

$$\frac{\partial I_{1n}}{\partial x} = -\frac{1}{\rho_x}\frac{\partial I_{1n}}{\partial m} = -\frac{1}{\rho_x \alpha}\left[\mathcal{H}\left(1+\frac{m}{\alpha}\right) - \mathcal{H}\left(\frac{m}{\alpha}\right)\right]\mathcal{H}\left(\frac{\alpha p - \beta m}{\alpha\beta}\right)\left(-\frac{m}{\alpha}\right)^n, \tag{A.64}$$

$$\frac{\partial I_{2n}}{\partial x} = -\frac{1}{\rho_x}\frac{\partial I_{2n}}{\partial m} = -\frac{1}{\rho_x \alpha}\left[\mathcal{H}\left(1+\frac{m}{\alpha}\right) - \mathcal{H}\left(\frac{m}{\alpha}\right)\right]\left(-\frac{m}{\alpha}\right)^n, \tag{A.65}$$

$$\begin{aligned}\frac{\partial I_{1n}}{\partial y} = -\frac{1}{\rho_y}\frac{\partial I_{1n}}{\partial p} = -\frac{1}{\rho_y \beta}\Bigg[&\mathcal{H}\left(\frac{m+\alpha}{\alpha}\right)\mathcal{H}\left(\frac{-\beta m + \alpha p}{\alpha\beta}\right) - \\ &+\mathcal{H}\left(\frac{m+\alpha}{\alpha}\right)\mathcal{H}\left(\frac{p+\beta}{\beta}\right) + \\ &-\mathcal{H}\left(\frac{-\beta m + \alpha p}{\alpha\beta}\right)\mathcal{H}\left(\frac{m}{\alpha}\right) + \mathcal{H}\left(\frac{m}{\alpha}\right)\mathcal{H}\left(\frac{p}{\beta}\right)\Bigg]\left(-\frac{p}{\beta}\right)^n,\end{aligned} \tag{A.66}$$

$$\frac{\partial I_{2n}}{\partial y} = -\frac{1}{\rho_y}\frac{\partial I_{2n}}{\partial p} = -\frac{1}{\rho_y \beta}\left[\mathcal{H}\left(1+\frac{p}{\beta}\right) - \mathcal{H}\left(\frac{p}{\beta}\right)\right]\left(-\frac{p}{\beta}\right)^n. \tag{A.67}$$

Appendix B

Arc Length Numerical Continuation

Let us consider the following non-linear system of equation

$$\mathbf{\Psi}(\mathbf{a}, \lambda) = \mathbf{R}(\mathbf{a}) - \lambda \mathbf{P} = 0 \tag{B.1}$$

where $\mathbf{a}$ is a $n \times 1$ vector and λ is a scalar. An equilibrium state $(\mathbf{a}_0, \lambda_0)$ satisfies a convergence criterion

$$\frac{|\Psi(\mathbf{a}_0, \lambda_0)|}{|\lambda_0 \mathbf{P}|} \leq \beta \tag{B.2}$$

where β is a predefined tolerance, typically set to 0.001.

Let us define

Iterative change δ With the symbol δ it is indicated the change between two successive iterations.

$$\delta \mathbf{a}_k = \mathbf{a}_{k+1} - \mathbf{a}_k. \tag{B.3}$$

Incremental change Δ With the symbol Δ it is indicated the change between an iteration $\mathbf{a}_k$ and the previous convergent state $\mathbf{a}_0$

$$\Delta \mathbf{a}_k = \mathbf{a}_k - \mathbf{a}_0, \tag{B.4a}$$

$$\Delta \lambda_k = \lambda_k - \lambda_0. \tag{B.4b}$$

Hence

$$\Delta \mathbf{a}_{k+1} = \Delta \mathbf{a}_k + \delta \mathbf{a}_k, \tag{B.5a}$$

$$\Delta \lambda_{k+1} = \Delta \lambda_k + \delta \lambda_k. \tag{B.5b}$$

Equation (B.1) is a system of n equations in $n+1$ unknowns, therefore an additional equation (usually called *constraint* equation) is required.

$$\boldsymbol{\Psi}(\mathbf{a}, \lambda) = \mathbf{R}(\mathbf{a}) - \lambda \mathbf{P} = 0, \tag{B.6a}$$

$$f(\Delta \mathbf{a}, \Delta \lambda) = \Delta \mathbf{a}^T \Delta \mathbf{a} + \alpha^2 \Delta \lambda^2 \mathbf{P}^T \mathbf{P} - ds^2 = 0. \tag{B.6b}$$

Equation (B.6b) is the *arc-length constraint*.

B.1 Continuation of Equilibrium

Continuation is essentially the task of getting a new equilibrium state $(\mathbf{a}_1, \lambda_1)$ starting from a previous convergent one $(\mathbf{a}_0, \lambda_0)$. Continuation is divided in two phases: *predictor* and *corrector*.

B.1.1 Predictor Phase

In the *predictor* phase, one moves from the equilibrium state $(\mathbf{a}_0, \lambda_0)$ to the first tentative state $(\mathbf{a}_1, \lambda_1)$. Expanding equation (B.1) in Taylor series around $(\mathbf{a}_0, \lambda_0)$

$$\boldsymbol{\Psi}(\mathbf{a}_1, \lambda_1) = \boldsymbol{\Psi}(\mathbf{a}_0, \lambda_0) + \frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{a}}(\mathbf{a}_0, \lambda_0)\delta \mathbf{a}_0 + \frac{\partial \boldsymbol{\Psi}}{\partial \lambda}(\mathbf{a}_0, \lambda_0)\delta \lambda_0 + \dots \tag{B.7}$$

since $\boldsymbol{\Psi}(\mathbf{a}_0, \lambda_0) = 0$ and supposing that $\boldsymbol{\Psi}(\mathbf{a}_1, \lambda_1) \cong= 0$[1] and neglecting higher order terms, equation (B.7) becomes

$$\mathbf{K}_T(\mathbf{a}_0, \lambda_0)\delta \mathbf{a}_0 - \mathbf{P}(\mathbf{a}_0, \lambda_0)\delta \lambda_0 = 0 \tag{B.8}$$

where $\dfrac{\partial \boldsymbol{\Psi}}{\partial \mathbf{a}}(\mathbf{a}_0, \lambda_0) = \mathbf{K}_T(\mathbf{a}_0, \lambda_0)$ is the *tangent stiffness matrix* at state $(\mathbf{a}_0, \lambda_0)$ and $\dfrac{\partial \boldsymbol{\Psi}}{\partial \lambda}(\mathbf{a}_0, \lambda_0) = -\mathbf{P}$. Mathematically, equation (B.8) is an undetermined system of equation, therefore there are *infinite* solutions

$$\begin{bmatrix} \delta \mathbf{a}_0^T & \delta \lambda_0 \end{bmatrix} \in \text{Null}\left(\begin{bmatrix} \mathbf{K}_T & -\mathbf{P} \end{bmatrix} \right). \tag{B.9}$$

[1] it is *hoped* that convergence is achieved in the next step

The constraint equation (B.6b) allows to choose the one of length ds.

$$f(\Delta\mathbf{a}_1, \Delta\lambda_1) = \Delta\mathbf{a}_1^T\Delta\mathbf{a}_1 + \psi^2\Delta\lambda_1^2\mathbf{P}^T\mathbf{P} - ds^2 = \delta\mathbf{a}_0^T\Delta\mathbf{a}_0 + \psi^2\Delta\lambda_0^2\mathbf{P}^T\mathbf{P} - ds^2. \tag{B.10}$$

Thus, from equation (B.8)

$$\delta\mathbf{a}_0 = \delta\lambda_0\mathbf{K}_T^{-1}\mathbf{P} = \delta\lambda_0\delta\mathbf{a}_P, \tag{B.11}$$

where $\delta\mathbf{a}_P = \mathbf{K}_T^{-1}\mathbf{P}$ and $\delta\lambda$ is defined from the equation (B.10)

$$(\delta\lambda_0\delta\mathbf{a}_P)^T(\delta\lambda_0\delta\mathbf{a}_P) + \psi^2\Delta\lambda_0^2\mathbf{P}^T\mathbf{P} = ds^2. \tag{B.12}$$

Therefore,

$$\delta\lambda_0 = \text{sgn}\,\frac{ds}{\sqrt{\delta\mathbf{a}_P^T\delta\mathbf{a}_P + \psi^2\mathbf{P}^T\mathbf{P}}} \tag{B.13}$$

where sgn $= \pm 1$.

There are different ways to solve the ambiguity on the sign in equation (B.13). For example, Crisfield (1991)

$$\text{sgn} = \begin{cases} 1 & \mathbf{a}_p^T\Delta\mathbf{a} + \psi^2\Delta\lambda \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{B.14}$$

where $\Delta\mathbf{a}$ and $\Delta\lambda$ are the increments with respect to the two previous convergent states at steps n and $n-1$.

$$\Delta\mathbf{a} = \mathbf{a}_n - \mathbf{a}_{n-1} \qquad \Delta\lambda = \lambda_n - \lambda_{n-1}. \tag{B.15}$$

For the first increment $n = 1$, the following can be used in place

$$\text{sgn} = \begin{cases} 1 & \mathbf{a}_p^T\mathbf{P} \geq 0 \\ -1 & \text{otherwise} \end{cases}. \tag{B.16}$$

At the end of the predictor phase the situation is the following:

$$\begin{cases} \mathbf{a}_1 = \mathbf{a}_0 + \delta\mathbf{a}_0 \\ \lambda_1 = \lambda_0 + \delta\lambda_0 \end{cases} \tag{B.17}$$

and the convergence is checked: if the criterion is verified

$$\frac{|\Psi(\mathbf{a}_1, \lambda_1)|}{|\lambda_1\mathbf{P}|} \leq \beta \tag{B.18}$$

then $(\mathbf{a}_1, \lambda_1)$ is a convergent state, otherwise it is necessary to perform one or more corrections. These corrections are carried out in the *corrector phase*.

B.1.2 Corrector Phase

The corrector phase includes all the corrections from a state $(\mathbf{a}_1, \lambda_1)$ that does not satisfy equation (B.18) to a state $(\mathbf{a}_1, \lambda_1)$ that does satisfy (B.18). This is normally achieved within few corrections unless some exceptions described in subsection 6.3.4. Considering the generalized system of equation

$$\tilde{\mathbf{\Psi}}(\mathbf{a}_k, \lambda_k) = \begin{bmatrix} \mathbf{\Psi}(\mathbf{a}_k, \lambda_k) \\ \\ f(\Delta\mathbf{a}_k, \Delta\lambda_k) \end{bmatrix}. \tag{B.19}$$

Expanding in Taylor series and neglecting terms of order higher than the first

$$\underbrace{\tilde{\mathbf{\Psi}}(\mathbf{a}_k + \delta_k, \lambda_k + \delta\lambda_k)}_{\approx 0} = \underbrace{\tilde{\mathbf{\Psi}}(\mathbf{a}_k, \lambda_k)}_{\neq 0} + \frac{\partial\tilde{\mathbf{\Psi}}}{\partial\mathbf{a}}(\mathbf{a}_k, \lambda_k)\delta\mathbf{a}_k + \frac{\partial\tilde{\mathbf{\Psi}}}{\partial\lambda}(\mathbf{a}_k, \lambda_k)\delta\lambda_k \tag{B.20}$$

where

$$\frac{\partial\tilde{\mathbf{\Psi}}}{\partial\mathbf{a}} = \begin{bmatrix} \dfrac{\partial\mathbf{\Psi}}{\partial\mathbf{a}} \\ \\ \dfrac{\partial f}{\partial\mathbf{a}} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_T \\ \\ 2\Delta\mathbf{a}_k^T \end{bmatrix}, \tag{B.21}$$

$$\frac{\partial\tilde{\mathbf{\Psi}}}{\partial\lambda} = \begin{bmatrix} \dfrac{\partial\mathbf{\Psi}}{\partial\lambda} \\ \\ \dfrac{\partial f}{\partial\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{P} \\ \\ 2\Delta\lambda_k\psi^2\mathbf{P}^T\mathbf{P} \end{bmatrix}. \tag{B.22}$$

Thus, the expanded *Newton-like* problem to solve is

$$\begin{bmatrix} \mathbf{K}_T \\ \\ 2\Delta\mathbf{a}_k^T \end{bmatrix} \delta\mathbf{a}_k + \begin{bmatrix} -\mathbf{P} \\ \\ 2\Delta\lambda_k\psi^2\mathbf{P}^T\mathbf{P} \end{bmatrix} \delta\lambda_k + \tilde{\mathbf{\Psi}}(\mathbf{a}_k, \lambda_k) = 0 \tag{B.23}$$

or, in matrix form

$$\begin{bmatrix} \mathbf{K}_T & -\mathbf{P} \\ \\ 2\Delta\mathbf{a}_k^T & 2\Delta\lambda_k\psi^2\mathbf{P}^T\mathbf{P} \end{bmatrix} \begin{bmatrix} \delta\mathbf{a}_k \\ \\ \delta\lambda_k \end{bmatrix} = -\tilde{\mathbf{\Psi}}(\mathbf{a}_k, \lambda_k) \tag{B.24}$$

which solved gives the iterations

$$\begin{bmatrix} \delta\mathbf{a}_k \\ \\ \delta\lambda_k \end{bmatrix} = - \begin{bmatrix} \mathbf{K}_T & -\mathbf{P} \\ \\ 2\Delta\mathbf{a}_k^T & 2\Delta\lambda_k\psi^2\mathbf{P}^T\mathbf{P} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{\Psi}(\mathbf{a}_k, \lambda_k) \\ \\ f(\mathbf{a}_k, \lambda_k) \end{bmatrix}. \tag{B.25}$$

Remark Since equation (B.25) is simply a generalization of the Newton-Raphson (NR) step for problem (B.20), it could seem straightforward to adapt an existing NR solver in order to get a numerical continuation code. Moreover, equation (B.25) is solvable even when $\det(\mathbf{K}_T) = 0$, i.e. *bifurcation points*, that have a physical meaning in mechanics (for example the buckling of a beam is a bifurcation problem). Unfortunately this is not the case. Although mathematically correct, (B.25) in its brutal form it is disadvantageous when it comes to large scale simulations. In fact for finite element discretization (and for meshfree as well), $\mathbf{K}_T$ is usually large, sparse, symmetric and banded, therefore easy to invert with an existing and well established numerical method like for example Gauss elimination, or *LU factorization*. Particularly for FE this feature of the tangent stiffness matrix is particularly advantageous since $\mathbf{K}_T$ is tri-diagonal, which is even simpler to invert and to store. Looking at equation (B.25), it is evident how such symmetry is lost, since the matrix required for the inversion is now *augmented* with a row and a column that are different. The immediate consequence of the loss of symmetry is that the tangent stiffness matrix $\mathbf{K}_T$ is more difficult to invert numerically.

This inconvenient is eliminated by using the Boltz' lemma (or ABCD lemma) Wriggers (2008)

Lemma B.1.1 ABCD Lemma *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *an invertible matrix,* $\forall\ \mathbf{b},\ \mathbf{c},\ \mathbf{d} \in \mathbb{R}^n$

$$\begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \\ \mathbf{a}^T & c \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & 0 \\ \\ 0 & 0 \end{bmatrix} + \frac{1}{\kappa} \begin{bmatrix} (\mathbf{A}^{-1}\mathbf{b})\mathbf{a}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{b} \\ \\ -\mathbf{a}^T\mathbf{A}^{-1} & 1 \end{bmatrix} \tag{B.26}$$

where $\kappa = c - \mathbf{a}^T(\mathbf{A}^{-1}\mathbf{b})$.

Applying the ABCD lemma B.1.1 to equation (B.25), it is possible to show that

$$\delta\lambda_k = -\frac{f(\mathbf{a}_k, \lambda_k)/2 + \Delta\mathbf{a}_k^T\delta\mathbf{a}_\psi}{\Delta\mathbf{a}_k^T\delta\mathbf{a}_P + \psi^2\Delta\lambda_k\mathbf{P}^T\mathbf{P}} \tag{B.27a}$$

$$\delta\mathbf{a}_k = \delta\mathbf{a}_\psi + \delta\lambda_k\delta\mathbf{a}_P \tag{B.27b}$$

where

$$\delta \mathbf{a}_P = \mathbf{K}_T^{-1}\mathbf{P}, \tag{B.28a}$$

$$\delta \mathbf{a}_\psi = \mathbf{K}_T^{-1}(-\mathbf{\Psi}(\mathbf{a}_k, \lambda_k)). \tag{B.28b}$$

Equations (B.27) show that it is possible to directly solve (B.25) without inverting the *augmented* matrix but rather using equations (B.28a) and (B.28b), that prescribe, as desired, only the inversion of matrix $\mathbf{K}_T(\mathbf{a}_k, \lambda_k)$. Moreover, (B.28a) and (B.28b) can be solved simultaneously, as they can be considered a linear system of equations with two right-hand sides

$$\begin{bmatrix}\delta \mathbf{a}_P & \delta \mathbf{a}_\psi\end{bmatrix} = \mathbf{K}_T^{-1}\begin{bmatrix}\mathbf{P} & (-\mathbf{\Psi}(\mathbf{a}_k, \lambda_k))\end{bmatrix} \tag{B.29}$$

this is particularly suitable for numerical routines for example based on Gaussian elimination, where multiple right-hand sides can be considered in the solution. Furthermore, some continuation methods prefer to perform the update of the stiffness matrix only at the beginning of the predictor phase, rather than at every corrector phase. Although it might be less accurate, this approach consents to save computing time, since the assembly of the tangent stiffness matrix can be sometimes time-consuming.

If the tangent stiffness matrix is calculated at beginning of the predictor phase (i.e. moving from the equilibrium state $(\mathbf{a}_0, \lambda_0)$), it could be advantageous to consider LU factorization

$$\mathbf{K}_T(\mathbf{a}_0, \lambda_0) = \mathbf{L}\mathbf{U} \tag{B.30}$$

where $\mathbf{L}$ and $\mathbf{U}$ are respectively a lower triangular matrix and a upper triangular matrix.

In this case equation (B.29) is not only solved just using only forward and backward substitutions, but since $\mathbf{L}$ and $\mathbf{U}$ are already available at this point of the calculation, they can be stored and subsequently re-used in every corrector phase. Moreover, for *sparse* matrices, LU factorization computational cost depends only on the number of the non-zero entries of the matrix rather than on its size. This is clearly advantageous for large scale simulations where matrices have sizes that far outnumber the amount of non-zero values. This approach could be

disadvantageous because more memory is required to store **L** and **U** and also it could be inaccurate in points of the equilibrium path where the determinant of $\mathbf{K}_T$ changes sign. This is the case for delamination, as seen in chapter 6. The equilibrium paths might have one or more bifurcation points (typically when a crack begins to propagate) and might have many *false instabilities* points due to not sufficient nodes density, that cause a quick change in the sign of $\det \mathbf{K}_T$.

Finally, the state $(\mathbf{a}_k, \lambda_k)$ can be updated

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \delta\mathbf{a}_k \tag{B.31a}$$

$$\Delta\mathbf{a}_{k+1} = \Delta\mathbf{a}_k + \delta\mathbf{a}_k \tag{B.31b}$$

$$\lambda_{k+1} = \lambda_k + \delta\lambda_k \tag{B.31c}$$

$$\Delta\lambda_{k+1} = \Delta\lambda_k + \delta\lambda_k \tag{B.31d}$$

$$\tag{B.31e}$$

and convergence checked

$$\frac{|\Psi(\mathbf{a}_{k+1}, \lambda_{k+1})|}{|\lambda_{k+1}\mathbf{P}|} \leq \beta. \tag{B.32}$$

B.2 Line Search algorithm

Let us suppose that the load level λ is fixed to a value $\bar{\lambda}$. The equation (B.1) is now

$$\mathbf{\Psi}(\mathbf{a}, \bar{\lambda}) = \mathbf{R}(\mathbf{a}) - \bar{\lambda}\mathbf{P} = 0 \tag{B.33}$$

and can be solved with a standard NR method using a NR iteration

$$\delta\mathbf{a}_k = -\mathbf{K}_T^{-1}\mathbf{\Psi}(\mathbf{a}_k, \bar{\lambda}) \tag{B.34}$$

then repeat equation (B.31a) until convergence (B.32) is achieved.

NR method is said to be a *local* convergent method, meaning that quadratic rate of convergence is assured whenever the starting guess solution is *close* to the actual solution. Practically not always is possible to choose or to know *a priori* a good guess solution. Moreover, when $\det(\mathbf{K}_T)$ is close to zero, the resulting NR step (B.34) can be so large that the following situation might happen:

$$|\mathbf{\Psi}(\mathbf{a}_{k+1}, \bar{\lambda})| > |\mathbf{\Psi}(\mathbf{a}_k, \bar{\lambda})|. \tag{B.35}$$

The principle behind the *line search* method is exactly the search for an adequate decrease of the norm of the residual, to prevent the situation in equation (B.35). In order to do so, problem (B.33) is reformulated as

$$F(\mathbf{a}, \bar{\lambda}) = \frac{1}{2}\mathbf{\Psi}(\mathbf{a}, \bar{\lambda})^T \mathbf{\Psi}(\mathbf{a}, \bar{\lambda}) = 0. \tag{B.36}$$

It can be observed that the NR step (B.34) is a descent path for $F(\mathbf{a}, \bar{\lambda})$. Indeed,

$$\begin{aligned} \nabla F^T \delta\mathbf{a}_k = \frac{\partial F}{\partial \mathbf{a}}(-\mathbf{K}_T^{-1}\mathbf{\Psi}(\mathbf{a}, \bar{\lambda})) = -\Psi(\mathbf{a}, \bar{\lambda})^T \frac{\partial \mathbf{\Psi}}{\partial \mathbf{a}} \mathbf{K}_T^{-1}\mathbf{\Psi}(\mathbf{a}, \bar{\lambda}) = \\ = -\Psi(\mathbf{a}, \bar{\lambda})^T \mathbf{K}_T \mathbf{K}_T^{-1}\mathbf{\Psi}(\mathbf{a}, \bar{\lambda}) = -\Psi(\mathbf{a}, \bar{\lambda})^T \mathbf{\Psi}(\mathbf{a}, \bar{\lambda}) \leq 0. \end{aligned} \tag{B.37}$$

This may appear in contradiction with equation (B.35), since it may seem that according to (B.37), a NR step can only proceed towards a reduction of the norm of the residual. It must be observed, though, that equation (B.37) is *local* and also that the actual NR step depends from the inverse of the tangent stiffness matrix (equation (B.34)). Therefore the resulting $\delta\mathbf{a}_k$, being too large, can overshoot far from the root.

A remedy is the *Line-Search* method LS and a detailed exposition can be found in Press *et al.* (1986). The idea is not to take the full NR step (B.34) but only a fraction η

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \eta\delta\mathbf{a}_k \tag{B.38}$$

where η is chosen such that the residual f is sufficiently decreased

$$f(\mathbf{a}_{k+1}) \leq f(\mathbf{a}_k) + \alpha\nabla f^T \eta\delta\mathbf{a}_k \tag{B.39}$$

where α is a number generally chosen as 0.0001. The actual value of η can be calculated by minimizing the function

$$g(\eta) = f(\mathbf{a}_k + \eta\delta\mathbf{a}_k) \tag{B.40}$$

where

$$\frac{\partial g}{\partial \eta} = \nabla f^T \delta\mathbf{a}_k. \tag{B.41}$$

The function $g(\eta)$ is not known *a priori*, but can be approximated with all the informations available at the first step, i.e.

$$g(0) = f(\mathbf{a}_k), \tag{B.42a}$$

$$g(1) = f(\mathbf{a}_{k+1}), \tag{B.42b}$$

$$\frac{\partial g}{\partial \eta}(0) = \mathbf{\Psi}^T(\mathbf{a}_k, \bar{\lambda})\mathbf{K}_T(-\mathbf{K}_T^{-1})\mathbf{\Psi}^T(\mathbf{a}_k, \bar{\lambda}) = -|\mathbf{\Psi}(\mathbf{a}_k, \bar{\lambda})|^2 = -f(\mathbf{a}_{k+1}, \bar{\lambda}). \tag{B.42c}$$

Thus, the function $g(\eta)$ can be approximated with a quadratic polynomial

$$g(\eta) \approx \left[g(1) - g(0) - \frac{\partial g}{\partial \eta}(0)\right]\eta^2 + \frac{\partial g}{\partial \eta}(0)\eta + g(0) \tag{B.43}$$

and minimizing

$$\eta = -\frac{\dfrac{\partial g}{\partial \eta}(0)}{2\left[g(1) - g(0) - \dfrac{\partial g}{\partial \eta}(0)\right]}. \tag{B.44}$$

The process described above is iterative, until condition (B.39) is met. In the subsequent steps, more informations become available (the two values of g), so the function g can be modelled as a cubic polynomial [1]. The same process can be applied at the numerical continuation. In fact

$$\lambda_{k+1} = \lambda_k + \eta\delta_k, \tag{B.45}$$

$$\delta\mathbf{a}_k = \delta\mathbf{a}_\Psi + \eta\delta\lambda_k\delta\mathbf{a}_P. \tag{B.46}$$

In this case

$$g(\eta) = f(\mathbf{a}_k + \delta\mathbf{a}_\Psi + \eta\delta\lambda_k\delta\mathbf{a}_P, \lambda_k + \eta\delta_k), \tag{B.47}$$

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial \mathbf{a}} \\ \\ \dfrac{\partial f}{\partial \lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_T\mathbf{\Psi}(\mathbf{a}, \lambda) \\ \\ -\mathbf{P}^T\mathbf{\Psi}(\mathbf{a}, \lambda) \end{bmatrix}. \tag{B.48}$$

[1] this is the reason for the name Cubic Line Search method

Using

$$g(0) = f(\mathbf{a}_k, \lambda_k), \tag{B.49}$$

and

$$g(1) = f(\mathbf{a}_{k+1}, \lambda_{k+1}), \tag{B.50}$$

$$\frac{\partial g}{\partial \eta} = \nabla f^T \underbrace{\begin{bmatrix} \delta\mathbf{a}_k \\ \\ \delta\lambda_k \end{bmatrix}}_{\textit{full NR iteration}} = \mathbf{\Psi}^T(\mathbf{a}_k, \lambda_k) \begin{bmatrix} \mathbf{K}_T & -\mathbf{P} \end{bmatrix} \begin{bmatrix} \delta\mathbf{a}_k \\ \\ \delta\lambda_k \end{bmatrix}. \tag{B.51}$$

Equations (B.48), (B.49), (B.50) and (B.51) allow adaptation of an existing LS routine to a numerical continuation code. In fact, it suffices to enter as input the values (B.49), (B.50) and (B.48) instead of (B.42).

References

Abdelaziz, Y. & Hamouine, A. (2008). A survey of the extended finite element. *Computers and Structures*, **86**, 1141–1151. 15

Alfano, G. & Crisfield, M. (2001). Finite element interface models for the delamination analysis of laminated composites: mechanical and computational issues. *Int. J. Numer. Meth. Engng*, **50**, 1701–1736. 141, 149, 172, 175

Anderson, T. (1991). *Fracture mechanics*. CRC Press. 21

Atluri, S. & Zhu, T. (1998). A new Meshless Local Petrov-Galerkin (MLPG) approach in computational mechanics. *Computational Mechanics*, **22**, 117–127. 35, 64, 65

Babuska, I. & Melenk, J. (1997). The partition of unity method. *International Journal for Numerical Methods in Engineering*, **40**, 727–758. 134

Babuška, I., Banerjee, U. & Osborn, J. (2003). Survey of meshless and generalized finite element methods: A unified approach. *Acta Numerica*, **12**, 1–125.

Bank, R. (1990). *PLTMG, a software package for solving elliptic partial differential equations: users' guide 6.0*. Society for Industrial and Applied Mathematics. 10

Barbieri, E. & Meo, M. (2008). Delamination modelling using an Element Free Galerkin approach. In *13th European Conference on Composite Materials ECCM-13, Stockholm, Sweden*. 62, 135

Barbieri, E. & Meo, M. (2009a). A meshfree penalty-based approach to delamination in composites. *Composites Science and Technology*, **69**, 2169–2177. 63, 135, 142

Barbieri, E. & Meo, M. (2009b). Evaluation of the integral terms in reproducing kernel methods. *Computer Methods in Applied Mechanics and Engineering*, **198**, 2485–2507. 23, 49, 204

Barbieri, E. & Meo, M. (submitted). Computation of the integral terms in reproducing kernel element methods. *Computer Methods in Applied Mechanics and Engineering*. 204

Barenblatt, G. (1962). Mathematical Theory of Equilibrium Cracks in Brittle Failure. *Advances in Applied Mechanics*, **7**. 137

Beissel, S. & Belytschko, T. (1996). Nodal integration of the element-free Galerkin method. *Computer Methods in Applied Mechanics and Engineering*, **139**, 49–74. 37

Beissel, S., Gerlach, C. & Johnson, G. (2006). Hypervelocity impact computations with finite elements and meshfree particles. *International Journal of Impact Engineering*, **33**, 80–90. 35

Belinha, J. & Dinis, L. (2006). Analysis of plates and laminates using the element-free Galerkin method. *Computers and Structures*, **84**, 1547–1559.

Belinha, J. & Dinis, L. (2007). Nonlinear analysis of plates and laminates using the element free Galerkin method. *Composite Structures*, **78**, 337–350. 35

Belytschko, T. & Fleming, M. (1999). Smoothing, enrichment and contact in the element-free Galerkin method. *Computers and Structures*, **71**, 173–195. 32, 56, 63

Belytschko, T. & Tabbara, M. (1996). Dynamic fracture using element-free Galerkin methods. *International Journal for Numerical Methods in Engineering*, **39**, 923–938. 27

Belytschko, T., Gu, L. & Lu, Y. (1994a). Fracture and crack growth by element-free Galerkin methods. *Modelling Simul. Mater. Sci. Eng*, **2**, 519–534. 25, 27, 52, 53

Belytschko, T., Lu, Y. & Gu, L. (1994b). Element-free Galerkin methods. *International Journal for Numerical Methods in Engineering*, **37**, 229–256. 22, 24, 25, 62, 134

Belytschko, T., Lu, Y. & Gu, L. (1995a). Crack propagation by element-free Galerkin methods. *Engineering Fracture Mechanics*, **51**, 295–315. 26

Belytschko, T., Lu, Y., Gu, L. & Tabbara, M. (1995b). Element-free Galerkin methods for static and dynamic fracture. *International Journal of Solids and Structures*, **32**, 2547–2570. 27

Belytschko, T., Krongauz, Y., Fleming, M., Organ, D. & Snm Liu, W. (1996a). Smoothing and accelerated computations in the element free Galerkin method. *Journal of Computational and Applied Mathematics*, **74**, 111–126. 27, 28, 50, 52, 54, 135

Belytschko, T., Krongauz, Y., Organ, D., Fleming, M. & Krysl, P. (1996b). Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, **139**, 3–47. 6, 14, 17, 23, 47, 56, 64, 72

Black, T. & Belytschko, T. (1999). Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, **45**, 601–620. 14, 134

Bolotin, V. (1996). Delaminations in composite structures: its origin, buckling, growth and stability. *Composites. Part B, Engineering*, **27**, 129–145. 132

Bolotin, V. (2001). Mechanics of delaminations in laminate composite structures. *Mechanics of Composite Materials*, **37**, 367–380. 132

Bordas, S., Rabczuk, T. & Zi, G. (2008). Three-dimensional crack initiation, propagation, branching and junction in non-linear materials by an extended meshfree method without asymptotic enrichment. *Engineering Fracture Mechanics*, **75**, 943–960.

Borg, R., Nilsson, L. & Simonsson, K. (2002). Modeling of delamination using a discretized cohesive zone and damage formulation. *Composites Science and Technology*, **62**, 1299–1314.

Borg, R., Nilsson, L. & Simonsson, K. (2004). Simulating DCB, ENF and MMB experiments using shell elements and a cohesive zone model. *Composites Science and Technology*, **64**, 269–278.

Borst, R., Remmers, J. & Needleman, A. (2006). Mesh-independent discrete numerical representations of cohesive-zone models. *Engineering Fracture Mechanics*, **73**, 160–177.

Breitkopf, P., Rassineux, A., Touzot, G. & Villon, P. (2000). Explicit form and efficient computation of MLS shape functions and their derivatives. *Int. J. Numer. Meth. Engng*, **48**, 466. 36, 83, 85

Brighenti, R. (2005). Application of the element-free Galerkin meshless method to 3-D fracture mechanics problems. *Engineering Fracture Mechanics*, **72**, 2808–2820.

Camanho, P. & Dávila, C. (2002). Mixed-mode decohesion finite elements for the simulation of delamination in composite materials. *NASA-Technical Paper*, **211737**. 133, 148, 173, 179

Camanho, P., Dávila, C. & Ambur, D. (2001). Numerical Simulation of Delamination Growth in Composite Materials. 133

Camanho, P., Dávila, C. & de Moura, M. (2003). Numerical Simulation of Mixed-Mode Progressive Delamination in Composite Materials. *Journal of Composite Materials*, **37**, 1415. 133, 148

Campbell, J., Vignjevic, R. & Libersky, L. (2000). A contact algorithm for smoothed particle hydrodynamics. *Computer Methods in Applied Mechanics and Engineering*, **184**, 49–65. 20

Carpinteri, A., Ferro, G. & Ventura, G. (2003). The partition of unity quadrature in element-free crack modelling. *Computers and Structures*, **81**, 1783–1794.

Cartwright, C., Oliveira, S. & Stewart, D. (2001). A parallel quadtree algorithm for efficient assembly of stiffness matrices in meshfree galerkin methods. In *Parallel and Distributed Processing Symposium., Proceedings 15th International*, 1194–1198. 73

Cartwright, C., Oliveira, S. & Stewart, D. (2007). Parallel support set searches for meshfree methods. *SIAM Journal on Scientific Computing*, **28**, 1318–1334.

Chen, J., Pan, C., Wu, C. & Liu, W. (1996). Reproducing Kernel Particle Methods for large deformation analysis of non-linear structures. *Computer Methods in Applied Mechanics and Engineering*, **139**, 195–227. 32, 33

Chen, J., Pan, C. & Wu, C. (1997). Large deformation analysis of rubber based on a reproducing kernel particle method. *Computational Mechanics*, **19**, 211–227. 34

Chen, J., Crisfield, M., Kinloch, A., Busso, E., Matthews, F. & Qiu, Y. (1999). Predicting Progressive Delamination of Composite Material Specimens via Interface Elements. *Mechanics of Advanced Materials and Structures*, **6**, 301–317. xiii, xv, 149, 175, 176, 177

Chen, J., Wu, C., Yoon, S. & You, Y. (2001). A stabilized conforming nodal integration for Galerkin mesh-free methods. *Int. J. Numer. Meth. Eng*, **50**, 435–466. 37

Chen, L.W.e., J.S. (2000). Meshless particle methods (special issue). *Computational Mechanics*, **25**, 99–317. 18

Chen, L.W.e., J.S. (2004). Meshless methods: Recent advances and new applications (special issue). *Computer Methods in Applied Mechanics and Engineering*, **193**, 933–1321. 18

Chen, X., Liu, G. & Lim, S. (2003). An element free Galerkin method for the free vibration analysis of composite laminates of complicated shape. *Composite Structures*, **59**, 279–289.

Chen, Y., Eskandarian, A., Oskard, M. & Lee, J. (2005). Meshless analysis of high-speed impact. *Theoretical and Applied Fracture Mechanics*, **44**, 201–207. 35

Clough, R. (1960). The finite element method in plane stress analysis. In *Proceedings, 2nd ASCE Conference on Electronic Computations, Pittsburgh, September 1960*, 960. 1

Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2001). *Introduction to algorithms*. The MIT press. 73

Crisfield, M. (1991). *Non-Linear Finite Element Analysis of Solids and Structures*. John Wiley & Sons Inc. 136, 137, 227

Cueto-Felgueroso, L., Colominas, I., Mosqueira, G., Navarrina, F. & Casteleiro, M. (2004). On the Galerkin formulation of the smoothed particle hydrodynamics method. *International journal for numerical methods in engineering*, **60**, 1475–1512. 20

Dávila, C., Camanho, P. & De Moura, M. (2001). Mixed-mode decohesion elements for analyses of progressive delamination. In *Proceedings of the 42nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Seattle, WA*. 179

De, S. & Bathe, K. (2000). The method of finite spheres. *Computational Mechanics*, **25**, 329–345. 35

De Berg, M., Cheong, O., Van Kreveld, M. & Overmars, M. (2008). *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc. 10

de Borst, R. (2003). Numerical aspects of cohesive-zone models. *Engineering Fracture Mechanics*, **70**, 1743–1757.

de Borst, R. & Remmers, J. (2006). Computational modelling of delamination. *Composites Science and Technology*, **66**, 713–722.

de Borst, R., Remmers, J. & Needleman, A. (2004). Computational aspects of cohesive zone models. *Advanced Fracture Mechanics for Life and Safety Assesments-Stockholm (Sweden)*.

Dilts, G. (1999). Moving-least-squares-particle hydrodynamics-I. Consistency and stability. *International Journal for Numerical Methods in Engineering*, **44**, 1115–1155. 20

Dilts, G. (2000). Moving least-squares particle hydrodynamics II: conservation and boundaries. *International Journal for Numerical Methods in Engineering*, **48**, 1503–1524. 20

Doblare, M., Cueto, E., Calvo, B., Martinez, M., Garcia, J. & Cegonino, J. (2005). On the employ of meshless methods in biomechanics. *Computer Methods in Applied Mechanics and Engineering*, **194**, 801–821. 7, 8

Dolbow, J. & Belytschko, T. (1998). An introduction to programming the meshless element free Galerkin method. *Archives of Computational Methods in Engineering*, **5**, 207–241. 72

Dolbow, J. & Belytschko, T. (1999). Numerical integration of the Galerkin weak form in meshfree methods. *Computational Mechanics*, **23**, 219–230. 13, 36, 64, 65, 167

Dolbow, J., Moës, N. & Belytschko, T. (2000). Discontinuous enrichment in finite elements with a partition of unity method. *Finite Elements in Analysis & Design*, **36**, 235–260. 134

Duarte, C. (1995). A review of some meshless methods to solve partial differential equations. *TICAM Report*, 95–06. 7, 17

Duarte, C. & Oden, J. (1996). Hp clouds–an hp meshless method. *Numerical Methods for Partial Differential Equations*, **12**, 673–705. 24, 28, 29, 35, 38, 56, 59

Duff, I., Erisman, A. & Reid, J. (1989). *Direct methods for sparse matrices*. Oxford University Press, USA. 12

Duflot, M. (2006). A meshless method with enriched weight functions for three-dimensional crack propagation. *Int J Numer Methods Eng*, **65**, 1970–2006. 37

Duflot, M. & Nguyen-Dang, H. (2002). A truly Meshless Galerkin Method based on a Moving Least Squares Quadrature [J]. *Communications in Numerical Methods in Engineering*, **18**, 1–9. 36

Duflot, M. & Nguyen-Dang, H. (2004a). A meshless method with enriched weight functions for fatigue crack growth. *Int. J. Numer. Meth. Engng*, **59**, 1945–1961. 37

Duflot, M. & Nguyen-Dang, H. (2004b). Fatigue crack growth analysis by an enriched meshless method. *Journal of Computational and Applied Mathematics*, **168**, 155–164.

Eskandarian, A., Chen, Y., Oskard, M. & Lee, J. (2003). Meshless analysis of fracture. Plasticity and impact. *Proceedings of IMECE*, **3**, 15–21. 35

Fasshauer, G. (2002). Matrix-free multilevel moving least-squares methods. *Approximation Theory, X St. Louis, MO*, 271–278. 35

Fasshauer, G. (2005). Dual bases and discrete reproducing kernels: a unified framework for RBF and MLS approximation. *Engineering Analysis with Boundary Elements*, **29**, 313–325.

Fasshauer, G. (2007). *Meshfree approximation methods with Matlab*. World Scientific Pub Co Inc. 73

Fasshauer, G. & Zhang, J. (2003). Recent results for moving least squares approximation. *Geometric Modeling and Computing*, 163–176. 35

Fernández-Méndez, S. & Huerta, A. (2004). Imposing essential boundary conditions in mesh-free methods. *Computer Methods in Applied Mechanics and Engineering*, **193**, 1257–1275. 14

Finkel, R. & Bentley, J. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica*, **4**, 1–9. 73

Fleming, M., Chu, Y., Moran, B. & Belytschko, T. (1997). Enriched element-free galerkin methods for crack tip fiels. *International Journal for Numerical Methods in Engineering*, **40**, 1483–1504. 29, 30

Freund, L. (1990). *Dynamic Fracture Mechanics*. Cambridge University Press. 27

Fries, T. & Matthies, H. (2004). Classification and overview of meshfree methods, Informatikbericht Nr. 2003-3, Scientific Computing Univ. 7, 18, 19, 41, 48, 51, 66, 69, 135

Gavete, L., Benito, J., Falcon, S. & Ruiz, A. (2000). Implementation of essential boundary conditions in a meshless method. *Communications in numerical methods in engineering*, **16**, 409–421. 14

Gingold, R. & Monaghan, J. (1977). Smoothed Particle Hydrodynamics: theory and application. *Monthly Notices of the Royal Astronomical Society*, **181**, 375–389. 22

Gleich, K. & Jackson, T. (2002). Center for Composite Manufacturing Final Report. 5

Griebel, M. & Schweitzer, M. (2002a). A Particle-Partition of Unity Method–Part II: Efficient Cover Construction and Reliable Integration. *SIAM Journal on Scientific Computing*, **23**, 1655–1682. 73

Griebel, M. & Schweitzer, M. (2002b). A particle-partition of unity method, Part II: Efficient cover construction and reliable integration. *SIAM J. Sci. Comput*, **23**, 1655–1682.

Gu, Y. (2005). Meshfree Methods and their comparisons. *International Journal of Computational Methods*, **2**, 477–515. 7

Guiamatsia, I., Falzon, B., Davies, G. & Iannucci, L. (2009). Element-Free Galerkin modelling of composite damage. *Composites Science and Technology*. 135

Günther, F. & Liu, W. (1998). Implementation of boundary conditions for meshless methods. *Computer Methods in Applied Mechanics and Engineering*, **163**, 205–230. 14

Han, W. & Meng, X. (2001). Error analysis of the reproducing kernel particle method. *Computer Methods in Applied Mechanics and Engineering*, **190**, 6157–6181.

Hashin, Z. (1980). Failure criteria for unidirectional fiber composites. *Journal of Applied Mechanics*, **47**, 329. 132

Hillerborg, A., Modeer, M., Petersson, P. *et al.* (1976). Analysis of crack formation and crack growth in concrete by means of fracture mechanics and finite elements. *Cement and Concrete Research*, **6**, 773–782. 137

Huerta, A. & Fernandez-Mendez, S. (2000). Enrichment and coupling of the finite element and meshless methods. *Int. J. Numer. Meth. Engng*, **48**, 1615–1636. 56

Huerta, A., Belytscko, T., Fernández-Méndez, S. & Rabczuk, T. (2004). *Chapter on Meshfree Methods*.

Idelsohn, S. & Oñate, E. (2006). To mesh or not to mesh. That is the question. *Computer Methods in Applied Mechanics and Engineering*, **195**, 4681–4696. 7, 37, 38, 72

Idelsohn, S., Oñate, E., Calvo, N. & Del Pin, F. (2003). The meshless finite element method. *International Journal for Numerical Methods in Engineering*, **58**, 893–912. 35

Jin, X., Li, G. & Aluru, N. (2001). On the equivalence between least-squares and kernel approximations in meshless methods. *Computer Modeling in Engineering and Sciences*, **2**, 447–462. 17

Karihaloo, B. & Xiao, Q. (2003). Modelling of stationary and growing cracks in FE framework without remeshing: a state-of-the-art review. *Computers and Structures*, **81**, 119–129. 15

Khosravifard, A. & Hematiyan, M. (2009). A new method for meshless integration in 2D and 3D Galerkin meshfree methods. *Engineering Analysis with Boundary Elements*, **34**, 30–40. 36

Klaas, O. & Shephard, M. (2000). Automatic generation of octree-based three-dimensional discretizations for partition of unity methods. *Computational Mechanics*, **25**, 296–304. 73

Klein, P., Foulk, J., Chen, E., Wimmer, S. & Gao, H. (2001). Physics-based modeling of brittle fracture: cohesive formulations and the application of meshfree methods. *Theoretical and Applied Fracture Mechanics*, **37**, 99–166. 137

Krysl, P. & Belytschko, T. (1997). Element-free Galerkin method: Convergence of the continuous and discontinuous shape functions. *Computer Methods in Applied Mechanics and Engineering*, **148**, 257–277. 30, 53

Krysl, P. & Belytschko, T. (1999). The Element Free Galerkin Method for Dynamic Propagation of Arbitrary 3-D Cracks. *Int. J. Numer. Meth. Engng*, **44**, 767–780. 31

Krysl, P. & Belytschko, T. (2001). ESFLIB: A library to compute the element free Galerkin shape functions. *Computer Methods in Applied Mechanics and Engineering*, **190**, 2181–2206. 66, 72, 74

Lancaster, P. & Salkauskas, K. (1981). Surfaces generated by moving least squares methods. *Mathematics of Computation*, **37**, 141–158. 22, 24, 50

Leitao, V. (2001). A meshless method for Kirchhoff plate bending problems. *Int. J. Numer. Meth. Engng*, **52**, 1107–1130.

Li, S. & Liu, W. (1996). Moving least-square reproducing kernel method Part II: Fourier analysis. *Computer Methods in Applied Mechanics and Engineering*, **139**, 159–193. 17, 24, 50

Li, S. & Liu, W. (1999a). Reproducing kernel hierarchical partition of unity Part I: Formulation and theory. *Int. J. Numer. Methods Eng*, **45**, 251–288.

Li, S. & Liu, W. (1999b). Reproducing kernel hierarchical partition of unity Part II: Applications. *Int. J. Numer. Methods Eng*, **45**, 289–317.

Li, S. & Liu, W. (2004a). *Meshfree Particle Methods*. Springer. 11

Li, S. & Liu, W. (2004b). *Meshfree particle methods*. Springer Verlag. 18

Li, S., Hao, W. & Liu, W. (2000a). Mesh-free simulations of shear banding in large deformation. *International Journal of Solids and Structures*, **37**, 7185–7206. 35

Li, S., Hao, W., Liu, W., Center, A.H.P.C.R. & of Minnesota, U. (2000b). Numerical simulations of large deformation of thin shell structures using meshfree methods. *Computational Mechanics*, **25**, 102–116. 34

Li, S., Lu, H., Han, W., Liu, W. & Simkins, D. (2004). Reproducing kernel element method Part II: Globally conforming Im/Cn hierarchies. *Computer Methods in Applied Mechanics and Engineering*, **193**, 953–987. 217

Libersky, L. & Petschek, A. (1990). Smooth particle hydrodynamics with strength of materials. *Proceedings of the next Free-Lagrange Conference Jackson Lake Lodge, Moran, Wyoming, June*. 20

Libersky, L., Petschek, A., Carney, T., Hipp, J. & Allahdadi, F. (1993). High strain Lagrangian hydrodynamics: a three-dimensional SPH code for dynamic material response. *Journal of Computational Physics*, **109**, 67–75. 20

Liew, K., Ng, T. & Wu, Y. (2002a). Meshfree method for large deformation analysis–a reproducing kernel particle approach. *Engineering Structures*, **24**, 543–551. 35

Liew, K., Wu, Y., Zou, G. & Ng, T. (2002b). Elasto-plasticity revisited: numerical analysis via reproducing kernel particle method and parametric quadratic programming. *Int. J. Numer. Meth. Engng*, **55**, 669–683.

Liu, G. (2003). *Mesh Free Methods: Moving Beyond the Finite Element Method*. CRC Press. 2, 6, 7, 8, 9, 18, 21, 65, 73

Liu, G. & Tu, Z. (2002a). An adaptive procedure based on background cells for meshless methods. *Computer Methods in Applied Mechanics and Engineering*, **191**, 1923–1943. 85

Liu, G. & Tu, Z. (2002b). An adaptive procedure based on background cells for meshless methods. *Computer Methods in Applied Mechanics and Engineering*, **191**, 1923–1943.

Liu, T.O.J.e., W.K.; Belytschko (1996). Meshless methods (special issue). *Computer Methods in Applied Mechanics and Engineering*, **139**, 1–400. 18

Liu, W. & Jun, S. (1998). Multiple-Scale Reproducing Kernel Particle Methods for Large Deformation Problems. *Int. J. Numer. Meth. Engng*, **41**, 1339–1362. 34

Liu, W., Jun, S. & Zhang, Y. (1995). Reproducing kernel particle methods. *International journal for numerical methods in fluids*, **20**, 1081–1106. 22, 134

Liu, W., Chen, Y., Uras, R. & Chang, C. (1996). Generalized multiple scale reproducing kernel particle methods. *Computer Methods in Applied Mechanics and Engineering*, **139**, 91–157. 22, 23, 34

Liu, W., Hao, W., Chen, Y., Jun, S. & Gosz, J. (1997a). Multiresolution reproducing kernel particle methods. *Computational Mechanics*, **20**, 295–309. 23, 34, 49, 206

Liu, W., Li, S. & Belytschko, T. (1997b). Moving least-square reproducing kernel methods (I) Methodology and convergence. *Computer Methods in Applied Mechanics and Engineering*, **143**, 113–154. 17, 22, 24, 50

Liu, W., Hao, S., Belytschko, T., Li, S. & Chang, C. (1999). Multiple scale meshfree methods for damage fracture and localization. *Computational Materials Science*, **16**, 197–205.

Liu, W., Han, W., Lu, H., Li, S. & Cao, J. (2004). Reproducing kernel element method. Part I: Theoretical formulation. *Computer Methods in Applied Mechanics and Engineering*, **193**, 933–951.

Lu, H., Li, S., Simkins, D., Kam Liu, W. & Cao, J. (2004). Reproducing kernel element method Part III: Generalized enrichment and applications. *Computer Methods in Applied Mechanics and Engineering*, **193**, 989–1011.

Lucy, L. (1977). A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, **82**, 1013–1024. 20

Macri, M., De, S. & Shephard, M. (2003). Hierarchical tree-based discretization for the method of finite spheres. *Computers and Structures*, **81**, 789–803. 73

Melenk, J. & Babuška, I. (1996). The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, **139**, 289–314. 24, 29, 56

Meo, M. & Thieulot, E. (2005). Delamination modeling in a double cantilever beam. *Composite Structures*, **71**, 429–434.

Moës, N. & Belytschko, T. (2002). Extended finite element method for cohesive crack growth. *Engineering Fracture Mechanics*, **69**, 813–833. 134

Moes, N., Dolbow, J. & Belytschko, T. (1999). A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, **46**, 131–150. 14, 134

Mohammadi, S. (2008). *Extended finite element method*. Oxford: Blackwell Publishing. 15

Monaghan, J. (1982). Why Particle Methods Work. *SIAM Journal on Scientific and Statistical Computing*, **3**, 422. 20, 47

Monaghan, J. (1988). An introduction to SPH. *Computer Physics Communications*, **48**, 89–96. 20, 47

Monaghan, J. (1992). Smoothed Particle Hydrodynamics. *Annual Reviews in Astronomy and Astrophysics*, **30**, 543–574. 20, 22, 47

Moore, A. (1991). A tutorial on kd-trees. Extract from PhD Thesis, available from http://www.cs.cmu.edu/*sim*awm/papers.html. 73

Muravin, B. & Turkel, E. (2006a). Multiple Crack Weight for Solution of Multiple Interacting Cracks by Meshless Numerical Methods. *Int. J. Numer. Meth. Engng*, **67**, 1146–1159. 37

Muravin, B. & Turkel, E. (2006b). Spiral weight for modeling cracks in meshless numerical methods. *Computational Mechanics*, **38**, 101–111. 37, 55

Nayroles, B., Touzot, G. & Villon, P. (1992). Generalizing the finite element method: Diffuse approximation and diffuse elements. *Computational Mechanics*, **10**, 307–318. 22

Needleman, A. (1987). A continuum model for void nucleation by inclusion debonding. *Journal of applied mechanics*, **54**, 525–531. 137, 138

Nguyen, V., Rabczuk, T., Bordas, S. & Duflot, M. (2008). Meshless methods: A review and computer implementation aspects. *Mathematics and Computers in Simulation*. 7, 18, 38, 57, 69, 72

Onate, E., Idelsohn, S., Zienkiewicz, O., Taylor, R. *et al.* (1996). A finite point method in computational mechanics. Applications to convective transport and fluid flow. *International Journal for Numerical Methods in Engineering*, **39**, 3839–3866. 35

Orifici, A., Herszberg, I. & Thomson, R. (2008). Review of methodologies for composite material modelling incorporating failure. *Composite Structures*, **86**, 194–210. 131

Orkisz, J. & Krok, J. (2008). On classification of the meshless methods. In *8th. World Congress on Computational Mechanics (WCCM8)*. 17

Pantano, A. & Averill, R. (2002). A penalty-based finite element interface technology. *Computers and Structures*, **80**, 1725–1748. 135

Pantano, A. & Averill, R. (2004). A mesh-independent interface technology for simulation of mixed-mode delamination growth. *International Journal of Solids and Structures*, **41**, 3809–3831. 135

Paris, P. & Erdogan, F. (1963). A Critical Analysis of Crack Propagation Laws. *J. Basic Eng. Trans. ASME Series D*, **85**, 528–534. 27

Ponthot, J. & Belytschko, T. (1998). Arbitrary Lagrangian-Eulerian formulation for element-free Galerkin method. *Computer Methods in Applied Mechanics and Engineering*, **152**, 19–46. 34

Preparata, F. & Shamos, M. (1985). *Computational geometry: an introduction*. Springer. 10

Press, W., Flannery, B., Teukolsky, S., Vetterling, W. *et al.* (1986). *Numerical recipes*. Cambridge University Press New York. 148, 232

Puck, A. & Schürmann, H. (2002). Failure analysis of FRP laminates by means of physically based phenomenological models. *Composites Science and Technology*, **62**, 1633–1662. 131, 132

Puso, M., Chen, J., Zywicz, E. & Elmer, W. (2008). Meshfree and finite element nodal integration methods. *International Journal for Numerical Methods in Engineering*, **74**, 416–446. 37

Quinlan, N., Basa, M. & Lastiwka, M. (2006). Truncation error in mesh-free particle methods. *Int. J. Numer. Meth. Engng*, **66**, 2064–2085.

Rabczuk, T. & Belytschko, T. (2005). Adaptivity for structured meshfree particle methods in 2D and 3D. *Int J Numer Methods Eng*, **63**, 1559–1582. 41, 85

Rabczuk, T. & Eibl, J. (2003). Simulation of high velocity concrete fragmentation using SPH/MLSPH. *Int. J. Numer. Meth. Engng*, **56**, 1421–1444. 20

Rabczuk, T., Belytschko, T. & Xiao, S. (2004). Stable particle methods based on Lagrangian kernels. *Computer Methods in Applied Mechanics and Engineering*, **193**, 1035–1063.

Rao, B. & Rahman, S. (2004). An enriched meshless method for non-linear fracture mechanics. *Int. J. Numer. Meth. Engng*, **59**, 197–223. 35

Remmers, J., Wells, G. & De Borst, R. (2002). Analysis of delamination growth with discontinuous solid-like shell elements. In *Proceedings of the Fifth Congress on Computational Mechanics (WCCM V). Vienna University of Technology, Vienna, Austria*.

Remmers, J., Borst, R. & Needleman, A. (2003a). A cohesive segments method for the simulation of crack growth. *Computational Mechanics*, **31**, 69–77. 134, 152

Remmers, J., Borst, R. & Needleman, A. (2003b). Simulation of fast crack growth using cohesive segments. In *VII International Conference on Computational Plasticity. COMPLAS*. 152

Remmers, J., de Borst, R. & Needleman, A. (2008). The simulation of dynamic crack propagation using the cohesive segments method. *Journal of the Mechanics and Physics of Solids*, **56**, 70–92. 135

Rice, J.R. (1968). A Path Independent Integral and the Approximate Analysis of Strain Concentration by Notches and Cracks. *Journal of Applied Mechanics*, **35**, 379–386. 25, 131, 141, 154

Samimi, M., van Dommelen, J. & Geers, M. (2009). An enriched cohesive zone model for delamination in brittle interfaces. *International Journal for Numerical Methods in Engineering*, **80**. 187

Schembri, P., Crane, D. & Reddy, J. (2004). A three-dimensional computational procedure for reproducing meshless methods and the finite element method. *Int. J. Numer. Meth. Engng*, **61**, 896–927.

Shaofan, L. & Liu, W. (2002). Meshfree and particle methods and their applications. *Applied Mechanics Review*, **55**, 1–34. 18

Simkins, D., Li, S., Lu, H. & Kam Liu, W. (2004). Reproducing kernel element method. Part IV: Globally compatible Cn ($n \geq 1$) triangular hierarchy. *Computer Methods in Applied Mechanics and Engineering*, **193**, 1013–1034.

Sukky Jun, T.B., Wing Kam Liu (1998). Explicit Reproducing Kernel Particle Methods for large deformation problems. *International Journal for Numerical Methods in Engineering*, **41**, 137–166. 34

Sukumar, N. & Wright, R. (2007). Overview and construction of meshfree basis functions: From moving least squares to entropy approximants. *Int. J. Numer. Meth. Engng*, **70**, 181–205.

Sukumar, N., Moran, B., Black, T. & Belytschko, T. (1997). An element-free Galerkin method for three-dimensional fracture mechanics. *Computational Mechanics*, **20**, 170–175. 31

Sukumar, N., Moran, B. & Belytschko, T. (1998). The natural element method in solid mechanics. *Int. J. Numer. Meth. Eng*, **43**, 839–887. 35

Sun, C. & Zheng, S. (1996). Delamination characteristics of double-cantilever beam and end-notched flexure composite specimens. *Composites Science and Technology*, **56**, 451–459.

Sun, Y., Hu, Y. & Liew, K. (2007). A mesh-free simulation of cracking and failure using the cohesive segments method. *International Journal of Engineering Science*, **45**, 541–553. 63, 135

Swegle, J., Hicks, D. & Attaway, S. (1995). Smoothed Particle Hydrodynamics Stability Analysis. *Journal of Computational Physics*, **116**, 123–134. 20

Tabarraei, A. & Sukumar, N. (2005). Adaptive computations on conforming quadtree meshes. *Finite Elements in Analysis & Design*, **41**, 686–702. 73

Turon, A. (2007). *Simulation of delamination in composites under quasi-static and fatigue loading using cohesive zone models*. Ph.D. thesis, Universitat de Girona. 149, 170, 196

Turon, A., Camanho, P., Costa, J. & Dávila, C. (2006). A damage model for the simulation of delamination in advanced composites under variable-mode loading. *Mechanics of Materials*, **38**, 1072–1089.

Tvergaard, V. (1990). Material failure by void growth to coalescence. *Advances in Applied Mechanics*, **27**, 83–151. 140

Tvergaard, V. & Hutchinson, J. (1992). The Relation Between Crack Growth Resistance and Fracture Process Parameters in Elastic–Plastic Solids. *Journal of the Mechanics and Physics of Solids*, **40**, 1377–1397.

Ventura, G., Xu, J. & Belytschko, T. (2002). A vector level set method and new discontinuity approximations for crack growth by EFG. *Int. J. Numer. Methods Eng*, **54**, 923–944. 32, 59

Vignjevic, R. & Campbell, J. (2009). Review of development of the Smooth Particle Hydrodynamics (SPH) method. *Predictive Modeling of Dynamic Processes*, 367–396. 20

Vignjevic, R., Campbell, J. & Libersky, L. (2000). A treatment of zero-energy modes in the smoothed particle hydrodynamics method. *Computer methods in Applied mechanics and Engineering*, **184**, 67–85. 20

Vignjevic, R., Reveles, J. & Campbell, J. (2006). SPH in a total Lagrangian formalism. *Computer Modeling in Engineering & Sciences*, **14**, 181–198. 20

Wells, G. & Sluys, L. (2001). A new method for modelling cohesive cracks using finite elements. *International Journal for Numerical Methods in Engineering*, **50**, 2667–2682. 134

Wells, G., de Borst, R. & Sluys, L. (2002). A consistent geometrically nonlinear approach for delamination. *Int. J. Numer. Meth. Eng*, **54**, 1333–1355. 152

Wriggers, P. (2008). *Nonlinear finite element methods*. Springer Verlag. 229

Yang, Q. & Cox, B. (2005). Cohesive models for damage evolution in laminated composites. *International Journal of Fracture*, **133**, 107–137.

You, Y., Chen, J. & Lu, H. (2003). Filters, reproducing kernel, and adaptive meshfree method. *Computational Mechanics*, **31**, 316–326. 92

Zhang, Z., Zhou, J., Wang, X., Zhang, Y. & Zhang, L. (2004). Investigations on reproducing kernel particle method enriched by partition of unity and visibility criterion. *Computational Mechanics*, **34**, 310–329.

Zhou, J., Wen, J., Zhang, H. & Zhang, L. (2003). A nodal integration and post-processing technique based on Voronoi diagram for Galerkin meshless methods. *Computer Methods in Applied Mechanics and Engineering*, **192**, 3831–3843.

Zhou, J., Wang, X., Zhang, Z. & Zhang, L. (2005). Explicit 3-D RKPM shape functions in terms of kernel function moments for accelerated computation. *Computer Methods in Applied Mechanics and Engineering*, **194**, 1027–1035. 36, 83

Zhu, T., Zhang, J. & Atluri, S. (1998). A local boundary integral equation (LBIE) method in computational mechanics, and a meshless discretization approach. *Computational Mechanics*, **21**, 223–235. 35

Zienckiewicz, O. & Taylor, R. (1994). The Finite Element Method. *Mc Graw Hill, London*. 62

Zou, Z., Reid, S. & Li, S. (2003). A continuum damage model for delaminations in laminated composites. *Journal of the Mechanics and Physics of Solids*, **51**, 333–356. 140